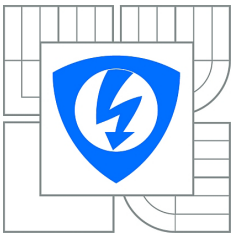
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘÍCÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

ŘÍZENÍ ROBOTU PRO SOUTĚŽ EUROBOT

CONTROL OF EUROBOT MOBILE ROBOT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

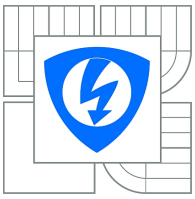
AUTOR PRÁCE
AUTHOR

LIBOR PLUCNAR

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. LUDĚK ŽALUD, Ph.D.

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student:	Libor Plucnar	***ID:***	125596
Ročník:	3	***Akademický rok:***	2011/2012

NÁZEV TÉMATU:

Řízení robotu pro soutěž EUROBOT

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se se soutěží EUROBOT, s minulými i současným zadáním soutěže. Navrhněte systém systém pro řízení tohoto robotu v průběhu soutěže. Zaměřte se především na strojové vidění, včetně návrhu použitého vybavení. Navržené algoritmy rovněž prakticky realizujte a otestujte.
Rezervováno - Libor Plucnar

DOPORUČENÁ LITERATURA:

Gordon McComb, Robot Builder's Bonanza, McGraw-Hill/TAB Electronics; 2 edition (September 21, 2000), ISBN-13: 978-0071362962

Termín zadání:	6.2.2012	***Termín odevzdání:***	28.5.2012

Vedoucí práce: doc. Ing. Luděk Žalud, Ph.D.
Konzultanti bakalářské práce:

doc. Ing. Václav Jirsík, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt
Bakalářská práce je zaměřena na teoretický vývoj řídícího algoritmu pro mobilní robot na základě soutěže EuRobot a jeho následnou praktickou realizaci dle konkrétního zadání. Jsou navrženy dvě hlavní metody řízení robotu dle vstupního signálu, podle kterého se robot orientuje. První způsob řízení je pomocí kamery. Tato část popisuje všeobecný postup vytvoření obrazového klasifikátoru, na základě kterého je možné vyhledávat objekty. Druhá část popisuje plánování trajektorie robotu v závislosti na jeho poloze tak, aby bylo dosaženo konkrétního bodu na předem určené mapě. Během řízení jsou zohledněny statické i dynamické překážky, které se mohou vyskytovat v prostředí, ve kterém se robot pohybuje. V rámci práce bylo navrženo také grafické ovládací rozhraní.

Summary
The thesis is focused on the theoretical development of a control algorithm for mobile robot based on Eurobot and its subsequent practical implementation according to a specific assignment. There are two main methods designed to control the robot according to the input signal, whereby the robot orientats. The first control method is using the camera. This section describes general procedure how to create the image classifier, under which it is possible to search objects. The second part describes the trajectory planning of the robot depends on its position to achieve a specific point on a predetermined map. During the proceedings are taken into account static and dynamic barriers that can occur in an environment in which the robot moves in. In the study was also designed graphical user interface.

Klíčová slova
EuRobot, Robot, Plánování trajektorie, Robotické vidění

Keywords
EuRobot, Robot, Path Planning, Robot Vision

PLUCNAR, L.*Řízení robotu pro soutěž Eurobot.* Brno: Vysoké učení technické v Brně, Fakulta elektroniky a informačních technologií, 2012. 53 s. Vedoucí doc. Ing. Luděk Žalud, Ph.D.

Prohlašuji, že svou bakalářskou práci na téma Řízení robotu pro soutěž EUROBOT jsem vypracoval bakalářskou pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: 28. května 2012

Libor Plucnar

Toto poděkování patří vedoucímu mé bakaláčské práce, a to doc. Ing. L. Žaludovi, Ph.D. za odbornou, metodickou i pedagogickou pomoc při zpracování mé bakalářské práce a v neposlední řadě za velice cenné rady a tipy při vlastním zhotovování.

Dále bych chtěl poděkovat svým kolegům M. Studenému a L. Podolanovi za jejich přínos a spolupráci na vývoji robotu.

V Brně dne: 28. května 2012

Libor Plucnar

Obsah

Seznam obrázků

1. Úvod

V rámci předchozího studia byla vytvořena pohybová platforma mobilního robotu založena na diferenciálním podvozku.Její další rozvoj by měl probíhat jak v oblasti řízení, tak v doplnění dalších periférií spolupracujících s řídící jednotkou. V konečné fázi by měl být robot schopen zcela autonomně řešit jednodušší problémy v rozsahu soutěže EuRobot. Je proto potřeba navrhnout a implementovat řídící mechanizmy schopné navigovat robota na základě různých datových signálů.

Jednou z hlavních oblastí, na kterou je potřeba se zaměřit je robotické vidění, soustavu zařízení a výpočetních mechanismů schopných detekovat zadané předměty, případně vytvořit synchronizaci mezi robotickým viděním a úchopným zařízením tak, aby bylo možné s objekty manipulovat.

2. Teoretický rozbor

2.1. Soutěž EuRobot

Soutěž EuRobot probíhá formou zápasu, kdy v jednom kole hrají proti sobě 2 týmy. Úkol pro oba týmy je shodný, pouze hřiště je zrcadlově obrácené, proto je třeba rozlišit, za který tým robot hraje. Každý tým se snaží dosáhnout během 90 sekund co největšího počtu bodů v zadané úloze. Obecně lze říci, že princip této soutěže nejčastěji vycházel z v nalezení, identifikaci a přemístění objektů na předem definované pozice během stanoveného časového limitu.

- **Chess'Up**(2011): roboti sbírali šachové figurky, vhodně je skládali do větších celků a rozmisťovali na své pole.
- **Feed the World**(2010): roboti sbírali předměty představující potraviny a přemisťovali je do připravených kontejnerů.
- **Temples of Atlantis**(2009): úkolem bylo ze stavebních prvků rozmístěných na ploše poskládat určené stavby.

Pro letošní rok 2012 bylo zvoleno téma soutěže **Ostrov pokladů**, jehož zadání se stalo hlavním podnětem k vyhotovení této bakalářské práce.

2.1.1. Cíle hry

Nalezení mapy k pokladu

Mapa k pokladu je reprezentována kusem látky. Pro získání bodů musí robot sebrat látku reprezentující mapu a mít ji až do konce hry u sebe. Mapa se nachází na předem určené pozici, která je pro všechny hry stejná.

Nalezení pokladu a jeho dopravení na „loď"

Robot má za úkol posbírat poklad rozmístěný na hrací ploše a dopravit jej do lodi. Předměty se budou nacházet nejen přímo na zemi, ale i v totemech uprostřed hrací plochy. Poklad bude zároveň možné ukrást z lodě soupeře.

Přečtení vzkazu v láhvi

Na delší straně hřiště se nachází 4 láhve se vzkazem (2 pro každý tým). Vzkaz robot přečte tím, že jej rozbalí stlačením tlačítka, které se nachází u každé lahve. Je třeba „otevřít" barvu svého týmu, jinak body získává soupeř.

2.1.2. Objekty

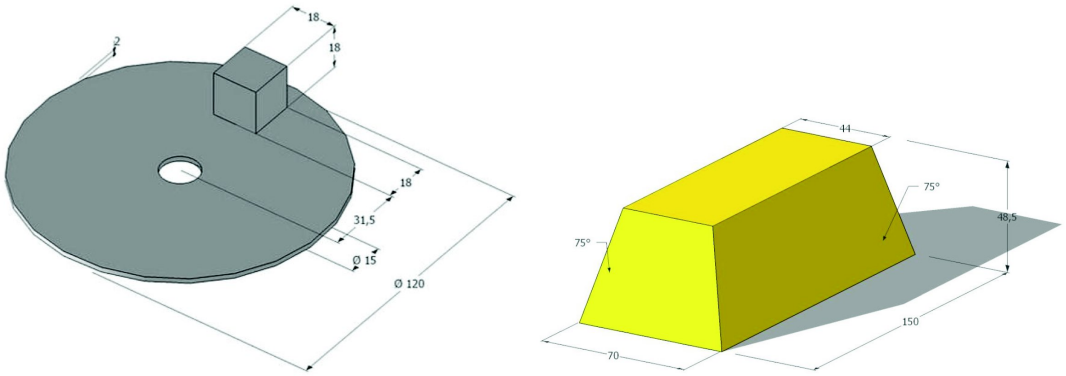
Mince (coins)

Mince jsou vyrobeny z disku CD, který je na jedné straně „podražený" krychlí o straně 18mm. Mincí jsou 2 druhy: černé a bílé. Černé mince není potřeba vyhledávat, protože

nejsou bodově ohodnoceny, a je možné se zaměřit pouze na bílé mince, které jsou ohodnoceny jedním bodem. Na hrací ploše je rozmístěno cca 38 černých a bílých mincí otočených podraženou stranou dolů. Technický nákres mince je na obrázku 2.1.

Cihličky (bullions)

Cihličky jsou vyrobeny ze dřeva a jsou žluté barvy. Boční stěny cihliček jsou zkoseny pod úhlem 75°. Na hrací ploše se nachází celkem 7 cihliček a každá je ohodnocena 3 body. Nákres cihličky je na obrázku 2.1.

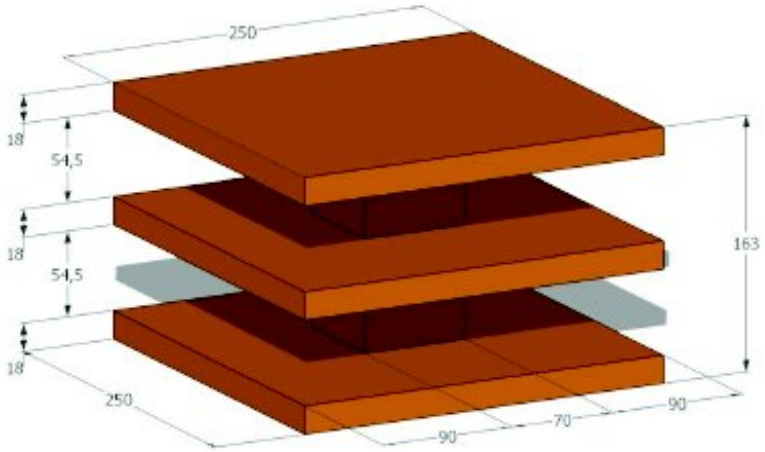


Obrázek 2.1: Bodované objekty: Mince (vlevo), Cihlička (vpravo) [9].

2.1.3. Mapa

Totem

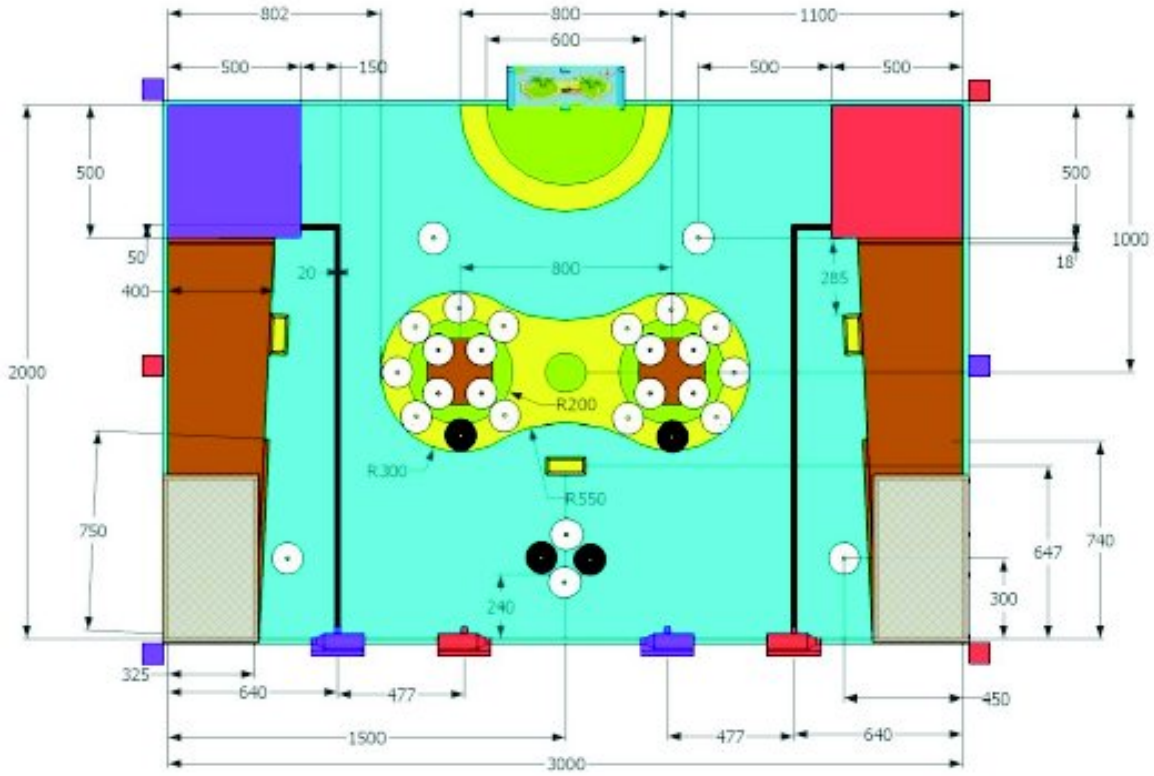
Totem je statická překážka. Na mapě se nacházejí celkem dva totemy ve středu mapy, každý se skládá ze tří pater, na prvním a posledním patře se nachází po 4 mincích, v prostředním patře se pak nacházejí 2 cihličky. Nákres totemu je na obrázku 2.2.



Obrázek 2.2: EuRobot: Totem [9].

Vykládací prostory - Lodě

Lodě se nacházejí při kratších stranách hřiště, pro každý tým je určena jedna loď. V horních rozích se nacházejí startovací plochy, ve kterých musí být robot umístěn před startem zápasu. Po skončení zápasu se počítájí objekty, které byly robotem dopraveny do lodi. Spodní část lodi je oddělena nízkou zídkou, přes kterou robot nemůže přejet.



Obrázek 2.3: EuRobot: Mapa hrací plochy [9].

Pravidla starších ročníků soutěže je možné nalézt na oficiálních stránkách soutěže [9]. Protože jsou stránky v současné době v rekonstrukci, pravidla pro rok 2012 se nacházejí v oficiálním fóru soutěže [10].

2.2. Snímání a zpracování obrazu

2.2.1. Robotické vidění [2]

Systémy robotického vidění mohou být jednoduché nebo složité dle specifických požadavků. Ty nejjednodušší umožňují pouhou detekci světla nebo jeho intenzity, složitější pak umožňují získat podrobnější informace o prostředí, ve kterém se robot pohybuje, nebo vnímat vzory, tvary, barvy a další informace o prostředí.

Přestože hardware pro sestavení očí robotu je spíše jednoduchý, s použitím vizuálních informací, které generují už to tak nebývá. S vyjímkou jednobuňkových světelných detektorů, musí být systém vídění propojen s počítačem, aby se stal použitelným.

Základní vizuální senzory

Mnoho jednoduchých zařízení může být použitých jako oči robotu. Tyto zahrnují následující:

- **Fotorezistory** jsou většinou buňky CdS sulfidu kadmia, které se chovají jako odpor závislý na světle. Odpor buněk se liší v závislosti na intenzitě dopadajícího světla. Pokud na buňku nedopadá žádné světlo, zařízení dosahuje velmi vysokých odporů typicky stovek KΩ až MΩ. Dopadající světlo tento odpor výrazně snižuje (o stovky až tisíc Ω). CdS buňky je snadné propojit s další elektronikou, ale reagují poměrně pomalu a nejsou schopny zaznamenat pokud se úroveň osvětlení změní 20x až 30x za sekundu. Na druhou stranu tato vlastnost příjde vhod, protože CdS buňky potlačují záblesky blikajícího světla napajeného střídavým proudem.

- **Fototranzistory** jsou podobné jako běžné tranzistory s odstraněným horním plastovým nebo kovovým krytem. Sklo nebo plastový kryt chrání citlivý tranzistorový substrát uvnitř. Narozdíl od CdS buněk fototranzistory reagují velice rychle a jsou schopny zaznamenat desítky až tisíce změn úrovně osvětlení za sekundu. Výstup fototranzistorů není lineární. Výstup fototranzistoru se neúměrně mění tím více, čím více světla na něj dopadá. Fototranzistor může být snadno oslepen velkým množstvím světla a není potom schopen detekovat další změny.

- **Fotodiody** toto jsou nejjednodušší diodové verze fototranzistorů. Jako fototranzistory jsou vyrobeny se skleněným nebo plastovým krytem který chrání polovodičový materiál uvnitř. Stejně jako fototranzistory reagují rychle na změny a mohou být oslepeny jsou-li vystaveny určité hranici osvětlení. Jednou společnou vlastností většiny fotodiod je, že jejich výstup je spíše nízký, ikdyž jsou vystaveny intenzivnímu osvětlení. To znamená, že výstup fotodiody musí být připojen k zesilovači, aby jejich použití bylo efektivní.

Zařízení citlivá na světlo se liší ve své spektrální oblasti, která je rozpětím viditelné a blízké infračervené oblasti elektromagnetického spektra, na které jsou také citlivé. CdS buňky vykazují spektrální odezvu podobnou lidskému oku s největší citlivostí na zelenou nebo žlutou barvu. Fototranzistory a fotodiody mají nejvyšší citlivost v infračervené a blízké infračervené oblasti. Navíc ve spolupráci s optickými filtry je možné snížit jejich citlivost na viditelné spektrum, a zvýšit tak jejich citlivost na infračervené a blízké infračervené světlo.

Zařízení mohou být buď jednobuňkové nebo mnohabuňkové v závislosti na počtu snímacích ploch. Jednobuňkové zařízení se používají převážně pro detekci přítomnosti světla (například optické brány). Mnohabuňkové zařízení se skládají z více jednobuňkových snímačů často spojených do řádků a sloupců, jejich výstupem je potom signálová matice, kterou lze použít k rozeznávání hrubých tvarů.

Video systémy v robotice

Pokud by bylo možné detekovat tvary objektů, robot by si mohl udělat předpokládaný obraz okolí, aby se v něm mohl pohybovat, případně rozpoznávat určité objekty.

Systémy pro robotické vidění nemusí být příliš sofistikované. Rozlišení obrazu může být třeba jen 100x100 pixelů (10 000 pixelů celkem), přesto je většinou upřednostňováno rozlišení větší než 300x200 pixelů (60 000 pixelů celkem). Větší rozlišení kamery znamená lepší schopnost rozpoznávat tvary.

Méně náročná práce bývá s video systémy, které přímo poskytují digitální výstup, než s těmi které mají pouze analogový výstup. Digitální video systémy je možné přímo připojit k počítači pomocí sériové, paralelní linky nebo USB. Analogové video systémy vyžadují kartu pro zaznamenávání videa, rychlý A/D převodník nebo podobné zařízení spojené s počítačem robotu.

I když v dnešní době je hardware pro robotické vidění relativně dostupný, překlad vizuálního obrazu pro robot vyžaduje rychlé zpracování a složité naprogramování. Objekt zkoumaný robotem může být nahlížen z mnoha úhlů, zkreslen dopadajícím světlem a stínem, a také se může objevovat v různých velikostech (v závislosti na vzdálenosti od předmětu), proto je potřeba všechny možnosti zobrazení zohlednit.

Obrázek 2.4: USB kamera pro snímání obrazu.

2.2.2. Knihovny pro zpracování obrazu

Aforge.net

Aforge.net je open source C framework navržený pro vývoj a výzkum na poli počítačového vidění a umělé inteligence, zpracování obrazu, neuronových sítí, genetických algoritmů, fuzzy logiky, strojového učení, robotiky a dalších. [7]

OpenCV (Open Computer Vision)

OpenCV je multiplatformní svobodná knihovna počítačového vidění s otevřeným vývojem zaměřená na zpracování obrazu a počítačové vidění. Knihovna také obsahuje funkce pro získávání obrazu přímo ze zařízení (webové kamery), GUI rozhraní pro jeho zobrazení, funkce pro ukládání obrazu do souborů a jejich opětovného načtení. Knihovna podporuje programovací jazyky C, C++, Python a další.

Jedním z cílů openCV je poskytnout snadno použitelnou infrastrukturu počítačového vidění, která by pomohla lidem vytvářet rychle velmi sofistikované aplikace počítačového vidění. Knihovna openCV obsahuje přes 500 funkcí, které zahrnují tovární kontrolu výrobků, lékařské zobrazování, bezpečnost, uživatelské rozhraní, kalibraci kamer, stereo vidění a robotiku. Protože počítačové vidění jde ruku v ruce se strojovým učením, openCV také obsahuje plnou, univerzální knihovnu strojového učení (Machine Learning Library – MLL). Tato podknihovna je zaměřena na statistické rozpoznávání a seskupování. [4]

VXL (the Vision - something - library)

Knihovna VXL je sbírka C++ knihoven navržených pro výzkum a implementaci počítačového vidění. Je napsána v ANSI/ISO C++ a je navržena tak, aby byla použitelná na mnoha platformách. Jádrové knihovny VXL jsou:

- **vnl (numeric)**: numerické funkce a algoritmy (matice, vektory, optimalizátory a další),
- **vil (imaging)**: načítání, ukládání a manipulace s obrázky v mnoha běžných formátech, zahrnuje také práci s velmi velkými obrázky,
- **vgl (geometry)**: geometrie pro body, křivky a další základní objekty v 1,2 nebo 3 dimenzích,
- **vsl (streaming I/O), vbl(basic templates), vul(utilities)**: různé, na platformě nezávislé funkce.

Kromě jádrových knihoven VXL obsahuje také knihovny pokrývající numerické algoritmy, zpracování obrazu, koordinaci systému, kamerovou geometrii, stereo, práci s videem, pravděpodobnostní modely, návrh GUI, topologii a další. [8]

2.2.3. Použité funkce knihovny openCV

cvCaptureFromCAM si bere jako argument číslo zařízení (pokud je zadáno -1, je použito první nalezené) a navrací odkaz na strukturu *CvCapture*. Tato struktura obsahuje všechny informace potřebné pro čtení dat z obrazového snímače.[4]

cvQueryFrame je kombinací funkcí *cvGrabFrame* a *cvRetrieveFrame*. Funkce *cvGrabFrame* získává snímek ze souboru/zařízení zadaného strukturou *CvCapture*. Tato funkce klade důraz na rychlost získání snímku, proto získaná data nejsou upravována a jsou uložena do interního bufferu, který je pro uživatele neviditelný. Jakmile je snímek uložen ve vstupním bufferu, je možné zavolat funkci *cvRetrieveFrame*, ta provádí se snímkem nezbytné operace a vrací jej jako ukazatel na objekt typu *IplImage*, který ukazuje na další

interní buffer. Snímek je uložen v této interní paměti a bude přepsán při zpracování dalšího snímku, proto je potřeba před jakoukoliv další manipulací s ním, jej překopírovat jinam. Funkce cvQueryFrame vrací objekt typu *Mat*, který je potom možné dále zpracovávat například pomocí kaskádových klasifikátorů, nebo *IplImage*.[4]

cvSetCaptureProperty a cvGetCaptureProperty umožňují nastavovat nebo získat informaci o parametrech snímků, čteného souboru nebo snímače obrazu, jako například rozlišení snímku (*CV_CAP_PROP_FRAME_WIDTH* - počet pixelů na výšku, *CV_CAP_PROP_FRAME_HEIGHT* - počet pixelů na šířku), pozici aktuálního snímku (*CV_CAP_PROP_POS_FRAME*; pro soubory videa) a další.[4]

cvSaveImage příjímá dva argumenty: jméno souboru, jehož koncovka udává formát, a odkaz na obrázek, který ukládá do zadaného souboru. Funkce vrací 1, pokud uložení bylo úspěšné nebo 0 v opačném případě.[4]

CascadeClassifier je třída pro detekci objektů, která pracuje s připravenými klasifikátory a na jejich základě je schopna rozpoznat objekty v obraze.[5]

CascadeClassifier::load načítá vygenerovaný klasifikátor ze souboru XML.[5]

CascadeClassifier::detectMultiScale vyhledává objekty podle klasifikátoru v zadaném obrazu. Jako parametry přijímá odkaz na obrázek, vektor, do kterého budou nalezené objekty uloženy, měřítko, parametry vyhledávání, minimální a maximální velikosti hledaných objektů.[5]

CascadeClassifier::empty udává, zda se podařilo načíst (nebo zda již byl načten) klasifikátor obrazu ze souboru XML.[5]

2.3. Plánování trajektorie

Fronta je abstraktní datová struktura typu FIFO (First In First Out: první vložený prvek bude ze seznamu první odstraněn). Frontu je možné si představit jako seznam hodnot, nad kterým jsou definovány následující operace:

- **vložení prvku** na konec seznamu,
- **vyzvednutí prvku** ze začátku seznamu,
- **dotaz na obsah**, kterým zjistíme, zda-li je seznam prázdný.

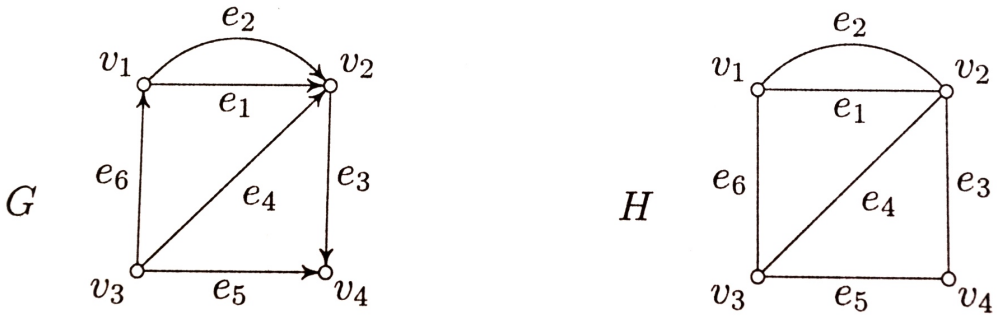
2.3.1. Grafy

Velká část algoritmů plánování trajektorie pohybu robotu v této práci (ale i částečně počítačové vidění) je postavena na teorii grafů, proto jsou zde uvedeny alespoň základní pojmy z této oblasti, které se vyskytnou v následujícím textu.

Graf se skládá z tzv. vrcholů a tzv. hran. Hrana vždy spojuje dva vrcholy a může být buď orientovaná, nebo neorientovaná. U hran orientovaných je rozlišován počáteční a koncový vrchol, a je možné říct, že hrana vede z počátečního do koncového vrcholu. Neorientované hrany chápeme jako symetrické spojení dvou vrcholů. [1]

Orientovaný graf je trojice $G = (V, E, \varepsilon)$ tvořená neprázdnou konečnou množinou V, jejíž prvky nazýváme vrcholy, konečnou množinou E, jejíž prvky se nazývají orientovanými hranami, a zobrazení $\epsilon : E \to V^2$, které nazýváme vztahem incidence. Toto zobrazení přiřazuje každé hraně $e \in E$ uspořádanou dvojici vrcholů (x, y). Prvý z nich x, je nazýván počátečním vrcholem hrany, druhý y, je nazýván koncovým vrcholem hrany. O hraně e je možné, říct že vede z vrcholu x do vrcholu y, a také, že spojuje vrcholy x a y. [1]

Neorientovaný graf je trojice $G = (V, E, \varepsilon)$ tvořená neprázdnou konečnou množinou V, jejíž prvky nazýváme vrcholy, konečnou množinou E, jejíž prvky jsou nazývány orientovanými hranami, a zobrazení $\epsilon : E \to V^2$, které je nazýváno vztahem incidence a které každé hraně $e \in E$ přiřazuje jednoprvkovou(smyčky) nebo dvouprvkovou množinu vrcholů. [1]



Obrázek 2.5: Orientovaný graf (G), Neorientovaný graf (H)[1].

Graf s implicitním zápisem hran je takový graf, u kterého není zápis hran na první pohled zřejmý. Existence hrany není přímo uvedena, ale je výsledkem např. matematické funkce. Příkladem takového grafu je halda uložená v jednorozměrném poli.

Graf s explicitním zápisem hran je takový graf, u kterého jsou hrany přímo uvedeny například v seznamu hran, v matici sousedností atd.

Sled je posloupnost vrcholů a hran $v_0, e_1, v_1, e_2, v_2, ..., v_k$, ve které platí, že každá hrana e_i spojuje vrcholy v_{i-1} a v_i. **Uzavřený sled** je takový sled, ve kterém se shodují počáteční a koncový vrchol. Pokud sled obsahuje každý vrchol maximálně jednou, bývá označován jako **cesta**, obsahuje-li sled každý vrchol maximálně jednou mimo první a poslední vrcholy, které jsou totožné, bývá označován jako **kružnice**.

Graf, ve kterém existuje kružnice se nazývá **cyklický graf**, graf ve kterém neexistuje žádná kružnice se nazývá **acyklický graf**.

Graf je souvislý, pokud každé jeho dva vrcholy můžeme spojit neorientovanou cestou.[1]

Strom je acyklický souvislý orientovaný graf. Každý vrchol stromu (mimo kořen) má právě jednoho rodiče (vrchol, z kterého do něj vede hrana) a může mít několik potomků. Strom kreslíme ve vrstvách tak, že v horní vrstvě se nachází kořen a v každé další vrstvě potomci vrstvy předcházející.

2.3.2. Prohledávání grafů

Základním účelem prohledávání grafů je zjišťování dostupnosti, tj. zjišťování, do kterých vrcholů grafu vedou cesty z daného výchozího vrcholu, případně také k nalezení těchto cest, tj. zjištění, kterými hranami a vrcholy procházejí. Dalším cílem prohledávání často bývá vykonání nějaké akce v každém dostupném vrcholu, popř. v každé dostupné hraně. [1]

Procházení grafu do hloubky

Procházení grafu do hloubky je možné si představit tak, že z počátečního vrcholu je graf procházen libovolným směrem, přitom jsou značeny vrcholy, které byly navštíveny. Je-li navštíven vrchol, ze kterého nevede žádná hrana nebo z něj nevede hrana do žádného doposud nenavštíveného vrcholu, prohledávání pokračuje ve vrcholu, který byl navštíven před tímto vrcholem. Takto je postupováno, dokud prohledávání neskončí v počátečním vrcholu a nelze z něj nadále pokračovat jinam.

Procházení grafu do šířky

Procházení grafu do šířky je možné si představit jako šířící se vlnu vypuštěnou z počátečního vrcholu všemi možnými směry. Nejprve jsou prohledávány vrcholy přímo spojené s počátečním vrcholem (vrcholy vzdálenosti 1 od počátečního vrcholu), poté se prohledávají nenavštívené vrcholy spojené přímo s těmito vrcholy atd. Každý vrchol je během prohledávání navštíven maximálně jednou. Prohledávání končí, neexistuje-li další vrchol, do kterého by se mohla vlna šířit.

2.3.3. Stavový automat [12]

Podle specifikace modelovacího jazyka UML jsou stavové automaty důležitou pomůckou při modelování dynamického chování systému. V průběhu průchodu stavy mohou být vykonávány různé aktivity. Diagramy stavových automatů podle této definice jsou tvořeny třemi základními prvky - stav, událost, přechod.

Charakteristika elementů, pro které má reprezentace stavovým automatem smysl:

- element reaguje na vnější události,
- životní cyklus elementu může být vyjádřen jako řada stavů, přechodů mezi nimi a událostí,

- chování elementu je důsledkem jeho předchozího chování.

Stav

Stav charakterizuje trvání konfigurace neměnných podmínek v systému. Stavem může být označená situace, kdy objekt čeká na událost nebo se objekt nějakým způsobem chová.

Každá akce ve stavu je přidružena k internímu přechodu, který je následkem události. Interní přechod umožňuje zachytit skutečnost významnou v rámci stavu, která ale nezpůsobuje přechod do stavu jiného.

Přechod

Přechod je přímé propojení jednotlivých stavů. Směřuje od zdrojového stavu k cílovému. Přechod ze stavu do stavu je reakcí na vznik události nebo ukončení činnosti (nebo činností) v aktivním stavu.

Událost

Specifikace UML definuje událost jako specifikaci něčeho významného, co se stane v určitém čase a prostoru, a co nemá trvání.

Stavový automat je možné znázornit jako orientovaný graf, kde vrcholy grafu znázorňují jednotlivé stavy, a hrany definují přechody mezi nimi. Může nastat situace, kdy jsou splněny podmínky pro přechod do dvou stavů zároveň. Z tohoto důvodu je možné přidat ohodnocení hran, které by jasně definovalo prioritu plnění jednotlivých podmínek.

2.4. Diferenciální podvozek

Pohybová platforma použita pro realizaci je založena na diferenciálním podvozku. Obsahuje 2 hnaná kola s enkodéry, pomocí kterých je možné regulovat jejich rychlost a také určit ujetou vzdálenost jednotlivých kol. Výstup s enkodéru má pulzní charakter, proto je jejich řízení vzorkováno s periodou $T_{vz} = 100ms$, během které dojde k přijetí až 20 pulzů (při maximální rychlosti), pro řízení je však použitá rychlost mnohem menší, aby bylo možné včasně reagovat na změnu pozice robotu a data snímaná ze senzorů. Se zmíněnou vzorkovací periodou také přichází zprávy o pohybu jednotlivých kol, pomocí kterých je možné určit, jak se změnila pozice robotu. Označíme-li dL počet pulzů detekovaných na levém kole a dR počet pulzů detekovaných na pravém kole za jednu vzorkovací periodu, bude potom změna úhlu natočení robotu [1]:

$$d\alpha_i = \frac{C_K(dL - dR)}{D} \quad [rad] \tag{2.1}$$

kde $C_K[cm]$ je konstanta motorů pro přepočet pulzů na vzdálenost (v cm/puls, protože pro pohyb robotu na mapě je zvolena centimetrová mřížka) a $D[cm]$ je rozchod kol robotu. Ujetá vzdálenost středu robotu dS bude potom:

$$dS_i = \frac{C_K(dL + dR)}{2} \quad [cm] \tag{2.2}$$

[1] za předpokladu použití dostatečně malé vzorkovací periody

Celkový úhel natočení robotu po n-tém vzorkování lze vyjádřit jako součet původního natočení α_0 (vzhledem k mřížce) a všech dílčích změn úhlů.

$$\alpha_n = \alpha_0 + \sum_{i=1}^{n} d\alpha_i \quad [rad] \tag{2.3}$$

Počáteční pozice robotu je dána souřadnicemi x_0 a y_0, pozice po n-tém vzorkování je potom:

$$x_n = x_{n-1} + dS_n.cos(\alpha_n) \quad [cm] \tag{2.4}$$

$$y_n = y_{n-1} + dS_n.sin(\alpha_n) \quad [cm] \tag{2.5}$$

Pozice je počítána s ohledem na zobrazování v obrázku tak, aby nebylo potřeba pro zobrazování provádět další přepočty za běhu programu, proto se nula nachází v levém horním rohu.

Podrobné odvození těchto vzorců je možno nalézt v [3].

3. Detekce objektů

Pro detekci objektů byla použita knihovna OpenCV. Postupy popsané v této části vycházejí z [4].

3.1. Obrazový klasifikátor

Pro detekci objektů je potřeba vygenerovat klasifikátor, konkrétně pro popisovanou metodu se jedná o Haar klasifikátor, založený na technice stromu. Tato technika detekce obrazu byla vyvinuta Paulem Violou a Michaelem Jonesem pro detekci obličejů a je známá jako Viola-Jonesův detektor.

Viola-Jonesův detektor používá formu AdaBoost, ale organizuje jej jako "zamítací kaskádu" uzlů, kde každý uzel je stromový AdaBoost klasifikátor navržený tak, aby měl vysokou úspěšnost detekce za cenu nízkého odmítacího hodnocení (hodně nesprávně detekovaných vzorů). Nenalezení objektu v kterémkoliv uzlu znamená, ukončení výpočtu na všech úrovních, což vede k negativnímu výsledku vyhledávání v dané oblasti. Objekt je v oblasti nalezen, pouze tehdy je-li nalezen na všech úrovních kaskády. [4]

3.2. Trénovací vzory

Pozitivní trénovací vzory

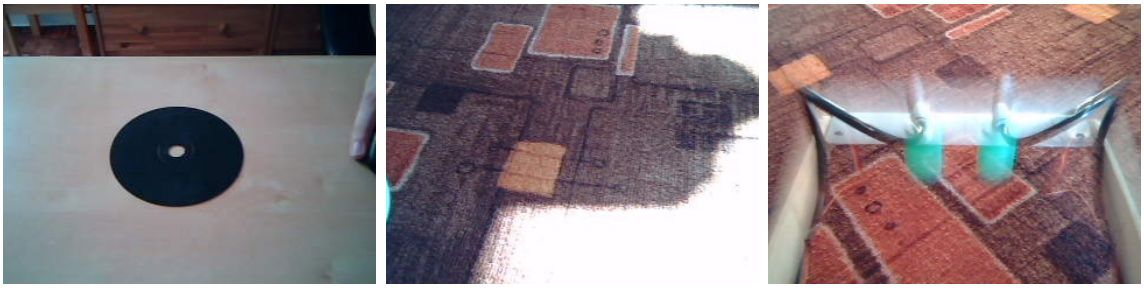
Pozitivní trénovací vzory jsou obrázky obsahující objekty, které je potřeba vyhledávat. U každého souboru je potřeba uvést, kde se v něm hledaný objekt nachází. Souřadnice objektu lze získat například v programu gimp nebo jiném grafickém editoru. Pro každý objekt jsou zapsány souřadnice levého horního rohu obdélníku, ve kterém se nachází, a také jeho výška a šířka. Velikost vzorů by neměla být menší než velikost nastavená pro předzpracování (viz dále).

Pozitivní trénovací vzory jsou uloženy v adresáři */positives/train*. Adresář obsahuje seznam souborů *positives.txt* s popisem pozic hledaných objektů, struktura souboru a jeho funkce bude rozebrána dále. Ukázka pozitivních trénovacích vzorů se nachází na obrázku 3.1.

Obrázek 3.1: Pozitivní trénovací vzory.

Negativní trénovací vzory

Negativní trénovací vzory jsou obrázky, které neobsahují hledané předměty. Např. obrázek pozadí, ve kterém se bude robot pohybovat. Velikost obrázků by neměla být menší než velikost nastavená pro základní rozměr vzoru. Negativní trénovací vzory se nacházejí v adresáři */negatives/train*. Adresář obsahuje seznam souborů *negatives.txt*. Ukázka negativních trénovacích vzorů se nachází na obrázku 3.2.



Obrázek 3.2: Negativní trénovací vzory.

Pozitivní testovací vzory

Pozitivní testovací vzory jsou takové obrázky, které obsahují předměty, které chceme vyhledávat. Na těchto vzorech je možné si ověřit, je-li míra naučení klasifikátoru dostatečná. Testovací vzory se nacházejí v adresáři */positives/testing*, kde je *i* seznam všech obrázků *testing.txt*, včetně popsaných pozic hledaných objektů.

3.3. Postup naučení systému

Program pro vyhledávání objektu v obraze je oddělený od klasifikátoru, který je generován během strojového učení. To umožňuje úpravu klasifikátoru bez nutnosti opakované kompilace programu nebo použití více různých klasifikátoru (např. pro různé objekty). Cílem učení je vygenerovat XML soubor klasifikátoru naučený na detekci požadovaného objektu v obraze. K tomu slouží programy z knihovny openCV opencv_createsamples, opencv_haartraining a opencv_performance pro OS linux nebo createsamples.exe, haartraining.exe a performance.exe pro Windows.

3.3.1. opencv_createsamples (createsamples.exe)

Tento program čte zadaný soubor (-info PATH), který obsahuje seznam pozitivních vzorů a pozice objektů v nich. Formát souboru je následující:

nazev_souboru.bmp n x0 y0 w0 h0 x1 y1 w1 h1 ...

Jako první parametr je cesta k obrázku, n je počet objektů, které se v obrázku nacházejí, x_i,y_i je pozice levého horního rohu v obdélníku obrázku, ve kterém se objekt nachází (pozice 0,0 odpovídá levému hornímu rohu obrázku), w_i a h_i jsou rozměry obdélníků vytyčujícího objekt. Program vytváří soubor vektorů (.vec), který obsahuje v hlavičce informace o velikosti vzorů (zadanou parametry $-w$ a $-h$) a jejich počtu a v těle zmenšeniny vzorů

na velikost základního vzoru. Díky tomuto předzpracování je možné provádět samotné učení v kratším čase.

Velikostí základního vzoru rozumíme nejmenším rozměrem objektu v obraze, který naučený klasifikátor bude rozpoznávat. Obdélník vytyčující objekt potom bude mít strany ve stejném poměru jako základní vzor.

Parametry programu:

```
-info $PATH   - cesta k seznamu souborů
-w $WIDTH     - výška předzpracovaných vzorů
-h $HEIGHT    - šířka předzpracovaných vzorůr
-num $NUMPOS  - počet pozitivních vzorů (počet souborů obrázků)
-vec $VEC     - cesta výstupnímu souboru vektorů
```

Příklad nastavení parametrů pro objekt mince: Vlivem perspektivy dochází ke zkreslení vyhledávaného objektů ve vertikální rovině (k jeho smrštění). Z pořízených obrazů je zřejmé, že parametr $WIDTH$ musí být větší než parametr $HEIGHT$. Experimentálně bylo zjištěno, že k nejlepším výsledkům dochází při hodnotách $WIDTH = 40$, $HEIGHT = 25$.

3.3.2. opencv_haartraining (haartraining.exe)

Tento program také vytváří soubor xml s naučeným systémem.

Parametry programu:

```
-vec $PATH            - předzpracovaný soubor vektorů
-bg $PATH             - cesta k seznamu souboru pozadí
-npos $NUMPOS         - počet pozitivních vzorů (předzpracovaných
                        v souboru $VEC)
-nneg $NUMNEG         - počet negativních vzorů (počet souborů)
-nstages $STAGES      - počet opakování (doporučuje se nastavit
                        14 - 25 opakování)
-mem $MEM             - velikost poskytnuté paměti RAM
-maxfalsealarm $MFA   - maximální hodnota falsealarm (viz dále)
-minhitrate $MHR      - minimální hodnota hitrate (viz dále)
-bt $ALG              - použitý algoritmus ( DAB, RAB, LB, GAB )
-w $WIDTH             - výška předzpracovaných vzorů
-h $HEIGHT            - šířka předzpracovaných vzorů
```

Protože se jedná o strojové učení, bude systém vždy dosahovat určité chyby (nepřesnosti). Tato chyba se skládá ze dvou složek:

- **Falsealarm** - poměr správně detekovaných negativních vzorů k celkovému počtu negativních vzorů během učení. Bude-li 200 negativních vzorů a z toho 50 bude detekováno chybně falsealarm bude $50/200 = 0{,}25$.
- **Hitrate** - poměr správně detekovaných pozitivních vzorů k celkovému počtu pozitivních vzorů. Bude-li 80 pozitivních vzorů, a z toho bude 40 detekováno správně, potom hitrate bude $40/80 = 0{,}5$.

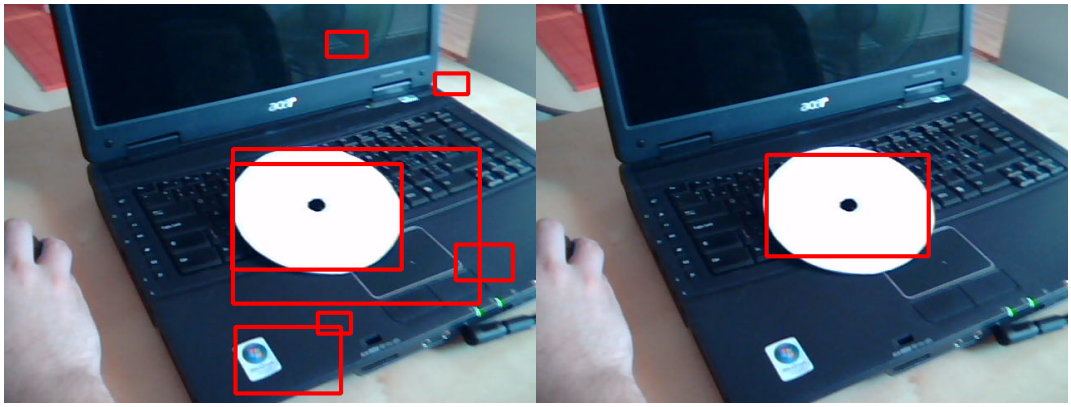
3.3.3. opencv_performance (performance.exe)

Tímto programem lze otestovat správnost naučení klasifikátoru. Jako vstup je seznam testovacích vzorů, včetně pozic hledaných objektů nacházejících se v obraze. Program kopíruje obrázky a označuje v nich nalezené objekty. Dále vytváří statistiky úspěšnosti nalezení nebo nenalezení objektů v jednotlivých obrazech a poskytuje informace o celkové úspěšnosti klasifikátoru při hledání.

Na obrázku 4.3 je příklad výstupu programu opencv_performance. Z prvního obrázku je vidět, že klasifikátor není dostatečně naučen. Míru naučení klasifikátoru lze ovlivnit přidáním učících vzorů, upravením velikosti základního vzoru nebo změnou počtu úrovní klasifikátoru $STAGE$.

Parametry programu:

```
-info $PATH   - seznam testovacích souborů
-w $WIDTH     - výška vzorů
-h $HEIGHT    - šířka vzorů
-rs $STAGES   - počet úrovní klasifikátoru
```



Obrázek 3.3: Kontrola míry naučení systému pomocí opencv_performance. Nedostatečně naučený systém s falešně pozitivními vzory (vlevo). Dostatečně naučený systém (vpravo).

3.4. Realizace

Pro řídící algoritmus byl vytvořen klasifikátor detekující objekty typu mince. Jako vzory byly použity snímky získané opakovaným testováním klasifikátoru v běžném provozu, během vývoje robotu. Byly tak neustále doplňovány nové vzory, vytvořené ze snímků z falešně pozitivním nebo falešně negativními nálezy, což dopomáhalo k eliminaci chyby klasifikátoru. Pro tento objekty byly zvoleny rozměry detekovaného vzoru 40x25. Objekty zaznačované v pozitivních trénovacích vzorech byly zaznačeny tak, že byla zvolena jejich pevná šířka zahrnující celý objekt a výška byla dopočtena tak, aby zachovávala poměr stran. Tímto způsobem se podařilo eliminovat geometrické zpoždění způsobené různou vzdáleností objektů od snímacího zařízení.

K vytvoření klasifikátoru byl napsán script ve skriptovacím jazyku BASH, který pracuje s popsanými soubory a poskytuje klasifikátor ve formátu XML. Tento script se nachází v příloze C. Poslední vygenerovaný obrazový klasifikátor byl vytvořen ze 45 pozitivních a 257 negativních vzorů, tato čísla však nemusí být konečná.

3.5. Nalezení a sledování objektů

Pro získání obrazu z kamery, rozpoznání a sledování objektů byla vytvořena knihovna *lib_opencv*, která je oddělena od řídícího programu robotu. Hlavním důvodem oddělení této knihovny od GUI rozhraní byla potřeba přiložení rozdílných knihoven při překladu programu. Navržené funkce vycházejí ze vzoru popsaného v [6]. V knihovně jsou implementovány následující funkce:

opencv_init() je funkce, která inicializuje knihovnu. Zajišťuje připojení webkamery a nastavení jejich parametrů (rozlišení snímaného obrazu). Načítá XML soubory obsahující použité klasifikátory a inicializuje strukturu *imgData*, která slouží k přenosu dat z kamery do GUI rozhraní, a pole objektů.

```
void opencv_init() {
    // pripojeni kamery
    capture = cvCaptureFromCAM(-1);

    if (!capture) {
        printf("Nepodarilo se pripojit kameru\n");
    } else {
        // nastaveni parametru kamery
        cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, CAPTURE_WIDTH);
        cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, CAPTURE_HEIGHT);
    }

    // Nacitani XML
    if (!coins_cascade.load("coins.xml")) {
        printf("--(!)Error loading coins.xml \n");
    }
    if (!bullions_cascade.load("bullions.xml")) {
        printf("--(!)Error loading bullions.xml \n");
    }

    // inicializace struktury pro obraz
    for (int i = 0; i < CAPTURE_WIDTH; i++)
        for (int j = 0; j < CAPTURE_HEIGHT; j++)
            imgData.red[i][j] = imgData.red[i][j] = imgData.red[i][j] = 0;

    //inicializace objektu
    for (int i = 0; i < MAX_OBJECTS; i++) {
        objects[i].type = NOOBJECT;
    }
}
```

opencv_getImage() je funkce, která zajišťuje získání snímku z webové kamery.

```
void opencv_getImage() {
```

```
    if (capture) {
        frame = cvQueryFrame(capture);

        if (frame.empty()) {
            printf(" --(!) No captured frame -- Break!\n");
            return;
        }
    }
}
```

opencv_saveImage() je funkce, která umožňuje ukládat snímky za běhu robotu. Tyto snímky je možné použít například pro úpravu klasifikátoru.

```
void opencv_saveImage() {
    char filename[50];
    printf("Saving image\n");

    // vytvoření adresáře pro snímky
    mkdir("images", S_IRWXU | S_IRWXG | S_IRWXO);
    //generování jména souboru
    sprintf(filename, "images/img%d.jpg", (int) time(NULL));

    IplImage *pSaveImg = cvQueryFrame(capture);
    cvSaveImage(filename, pSaveImg);
    printf("Image saved:%s\n", filename);
}
```

opencv_getObjects() je funkce, která v obraze získaném pomoci *opencv_getImage()* detekuje objekty a ukládá je ve struktuře *object*, vytvořené k ukládání nalezených objektů. Návratovou hodnotou této funkce je odkaz na první prvek pole struktur *object*. Tímto způsobem získá GUI rozhraní seznam detekovaných objektů.

```
object * opencv_getObjects() {

    if (frame.empty()) { // nepodarilo se ziskat snimek
        return objects ;
    }

    opencv_updateObjects();

    // detekce minci
    if (!coins_cascade.empty()) {
        std::vector<Rect> coins;

        // detekce objektu
        coins_cascade.detectMultiScale(frame, coins, 1.1, 2,
                                       0 | CV_HAAR_SCALE_IMAGE,
                                       Size(85,53), Size(150,94));

        for (int i = 0; i < coins.size(); i++) {
            opencv_addObject(coins[i].x, coins[i].y,
                             coins[i].width, coins[i].height, COIN);
        }
    }
    // detekce cihlicek
```

```
    if (!bullions_cascade.empty()) {
      ...
    }

    return objects;
}
```

Struktura object

```
typedef struct {
    int x, y;         // pozice objektu v obraze
    int w, h;         // rozmery objektu
    int ttl;          // Time To Live
    bool focus;       // Zemereni na objekt
    bool disabled;    // objekt byl zakázán
    object_type type; // typ objektu NOOBJECT, COIN, BULLION
} object;
```

Hodnoty x,y,w,h uchovávají informaci o umístění objektu v obraze, hodnota *ttl* udává životnost objektů (bude popsána podrobněji později), hodnota *type* určuje, jedná-li se o minci (COIN), cihličku (BULLION), nebo prázdnou položku (NOOBJECT). Všechny nalezené objekty (mince a cihličky) jsou skládány od začátku pole bez prázdných míst (NOOBJECT). Pokud by se objekt nacházel v lodi nebo mimo mapu, může být zakázán pomocí hodnoty *disabled*.

opencv_addObject() je funkce zajišťující ukládání nalezených objektů do pole objektů. Protože detekce objektů je vykonávaná zvlášť pro jednotlivé snímky a je potřeba uchovat informaci o objektu (hlavně z hlediska uchování informace o aktivitě objektu $focus$), je třeba objekty pro každý snímek spárovat. Přidávání objektů funguje následujícím způsobem: pokud existuje v poli objektů objekt, který měl na minulém snímku podobnou pozici (rozdíl pozic nepřesáhl maximální možnou odchylku a zároveň rozdíl rozměrů nepřesáhl maximální možnou odchylku), předpokládá se, že se jedná o ten samý objekt, obnovíme proto jeho životnost *ttl* na maximální hodnotu. Pokud podobný objekt nebyl nalezen, předpokládá se, že byl detekován poprvé, a je přidán do pole objektů. Pokud by se stalo, že se objekt posunul o větší než tolerovanou odchylku, vznikne sice objekt nový, původní objekt však po vypršení *ttl* zanikne.

Tato funkce také kontroluje, zda-li souhlasí rozměry objektů, touto kontrolou je možné eliminovat velké množství chyb klasifikátoru (ty příležitostně generují falešně pozitivní nálezy, často mnohem měnších rozměrů, než jakých může objekt při použitém nastavení kamery dosáhnout). Objekty nacházející se na vrcholu totemu budou touto kontrolu také eliminovány, protože v současné době konstrukční řešení robotu neumožňuje jejich bezpečné uchopení.

```
void opencv_addObject(int x, int y, int w, int h, object_type type) {
    int i;

    // kontrola platnosti objektu dle rozmeru
    if ( type == COIN && (w > 150 || w < 85))return;

    for (i = 0; i < MAX_OBJECTS - 1; i++) {
        if (objects[i].type == NOOBJECT) {
            // pridavame novy objekt
```

```
            objects[i].x = x;
            objects[i].y = y;
            objects[i].w = w;
            objects[i].h = h;
            objects[i].type = type;
            objects[i].focus = false;
            objects[i].disabled = false;
            objects[i].ttl = INIT_TTL;
        }
        // objekty se neshoduji
        if (objects[i].type != type)continue;

        if ((abs(objects[i].x - x) + abs(objects[i].y - y)) > 40)
            continue; // pozice objektu nesouhlasi
        if ((abs(objects[i].w - w) + abs(objects[i].h - h)) > 40)
            continue; // pozice objektu nesouhlasi

        objects[i].ttl = INIT_TTL;
        objects[i].x = x;
        objects[i].y = y;
        objects[i].w = w;
        objects[i].h = h;

        break;
    }
}
```

opencv_updateObjects() je funkce, která udržuje informace o detekovaných objektech aktuální pomocí jejich hodnoty *ttl*. Funkce poustupně projde všechny objekty a dekrementuje jejich životnost. Pokud životnost některého prvku skončí, přesune na jeho místo poslední objekt v poli. V poli takto nevznikají mezery a jeho průběžné procházení je mnohem rychlejší, protože jej lze ukončit, jakmile algoritmus narazí na objekt typu NOOBJECT.

```
void opencv_updateObjects() {
    int i, n;

    for (i = 0; i < MAX_OBJECTS; i++) {
        if (objects[i].type == NOOBJECT)break;
    }

    n = i;

    for (i = 0; i < MAX_OBJECTS; i++) {
        if (objects[i].type == NOOBJECT)break;

        objects[i].ttl--;

        if (objects[i].ttl < 0) {
            objects[i] = objects[n - 1];
            objects[n - 1].type = NOOBJECT;
            n--;
        }
    }
}
```

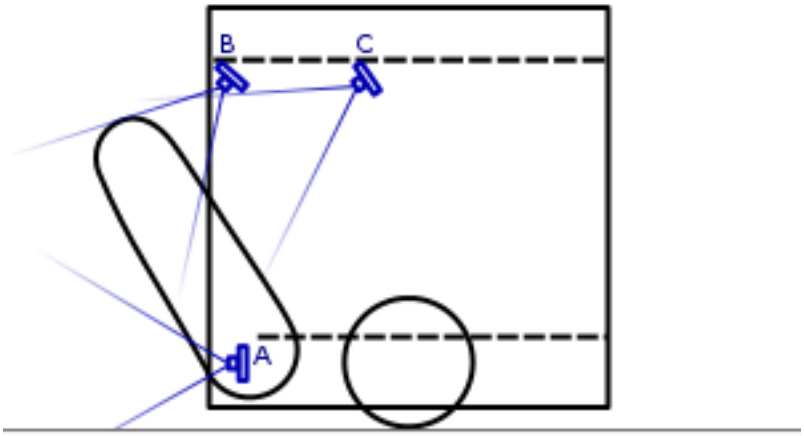
opencv_getFocusedObject() je funkce, která vybírá objekt, na který se robot zaměří a pokusí se jej získat. Funkce také zajišťuje změnu vybraného objektu, v případě, dostane-li se do zorného úhlu robotu jiný hledaný objekt, který by se nacházel blíže robotu a jeho sebrání by bylo výhodnější než sebrání vzdálenějšího objektu a návratu k původně bližšímu objektu. K převzetí objektu nemůže dojít, pokud by robot detekoval objekty přibližně ve stejné vzdálenosti, protože by mohlo docházet k situaci, kdy by robot neustále přepínal mezi dvěma či více objekty a nedošlo by tím pádem k dosažení žádného z nich. Proto je pro převzetí nezbytně nutné, aby vybraný objekt byl výrazně blíž než objekt, na který je robot zaměřen.

```
object * opencv_getFocusedObject() {
    int max_w = 0;
    int max_wi = -1;
    int i;

    object * t_object = NULL;

    // neexistuje zadny objekt
    if (objects[0].type == NOOBJECT)
        return NULL;

    for (i = 0; i < MAX_OBJECTS; i++) {
        if (objects[i].type == NOOBJECT)break;

        if (objects[i] .focus) t_object = &objects[i];

        if (objects[i].w > max_w && !objects[i].disabled) {
            max_w = objects[i].w;
            max_wi = i;
        }
    }

    if ( max_wi == -1 )return t_object;

    // focus nebyl pridelen
    if (t_object == NULL) {
        objects[max_wi].focus = true;
        t_object = &objects[max_wi];

        // existuje lepsi reseni?
    } else {
        if (objects[max_wi].w - t_object->w > 10) { // je vyhodne prevzit objekt?
            t_object->focus = false;
            t_object = &objects[max_wi];
            t_object->focus = true;
        }
    }
    return t_object;
}
```

opencv_resetObjects() je funkce, která maže všechny nalezené objekty z pole objektů. Je využita hlavně tehdy, přechází-li robot ze stavu, ve kterém nevyužívá navigaci pomocí kamery, do stavu řízeného kamerou. V poli zůstávají objekty z posledního snímání, přestože se pozice robotu změnila.

opencv_getImageData() je funkce, která slouží k přenesení obrazových dat z kamery do grafického uživatelského rozhraní rozhraní.

3.6. Umístění kamery



Obrázek 3.4: Možné umístění kamery na robotu, **A**-mezi rameny, **B**-pod horní nosnou plochou, **C**-pod horní nosnou plochou, zanořená.

Pozice kamery na robotu bylo potřeba zvolit tak, aby bylo možné snímat co největší prostor před robotem a zároveň aby ve snímaném obraze bylo možné vytyčit co největší cílovou oblast. Cílovou oblastí je myšlena pozice hledaného předmětu vůči robotu, ve které je možné předmět sebrat pomocí ramene. Tato oblast leží na předozadní ose robotu, proto je vhodné na tuto osu umístit také kameru. S ohledem na konstrukci byly zvoleny tři pozice pro umístění kamery (viz obrázek 3.4).

Pozice **A** odpovídá umístění kamery na podlahu konstrukce robotu. Výhodou této pozice je velký záběr prostoru před robotem. Nevýhodou této pozice je značné geometrické zkreslení objektu mince, pro který by bylo nutné vytvořit složitější klasifikátor uvažující také pohled ze všech stran objektu. Objekty by při nevhodném postavení mohly dokonce splývat s pozadím a bylo by obtížné je detekovat. Další nevýhodou je malá výška cílové oblasti a tím i obtížnější řízení.

Pozice kamery **B** odstraňuje problém s geometrickým zkreslením a nárůstem složitosti klasifikátoru, snímá však jen malý prostor před robotem a robot se stává "krátkozrakým", protože může detekovat pouze předměty v jeho těsné blízkosti. Tento problém by bylo možné eliminovat použitím širokoúhlého objektivu nebo vhodné optické soustavy, došlo by pak ale ke geometrickému zkreslení snímané oblasti. Toto zkreslení by bylo možné opravit softwarově za cenu většího časového vytížení procesoru a prodloužení doby detekce objektů.

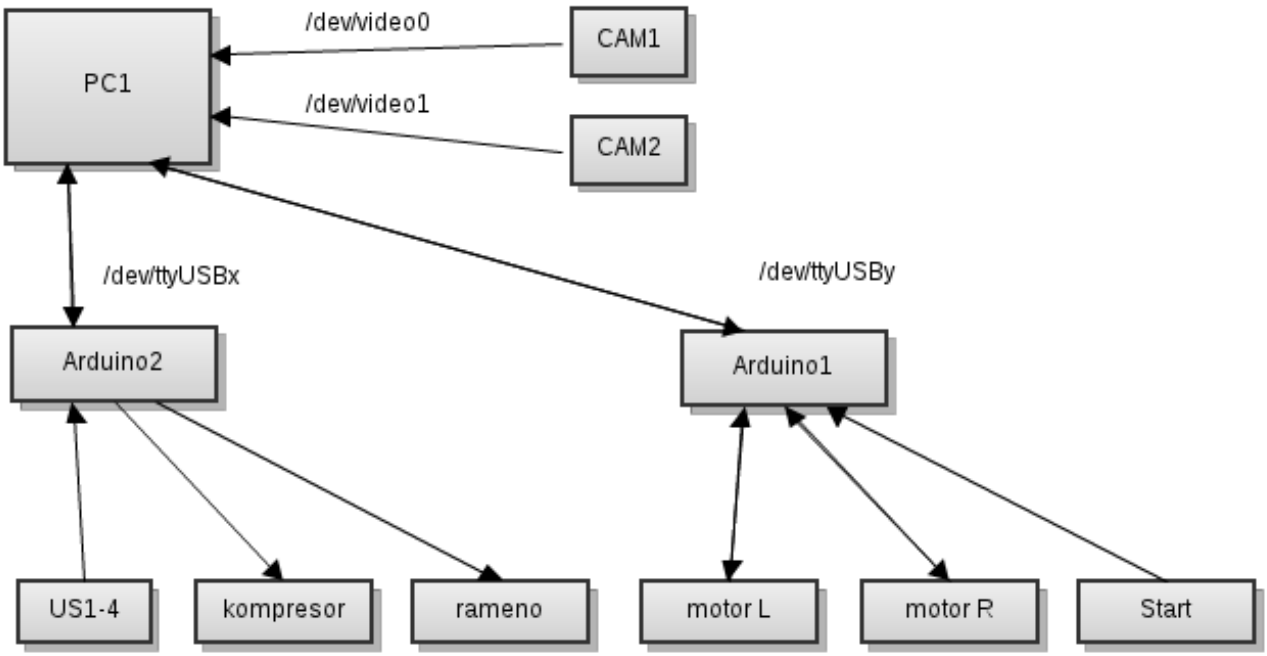
Pozice kamery **C** řeší jak problém s geometrickém zkreslením objektů, tak omezení snímané oblasti před robotem. Kamera je směrována tak, aby se cílová oblast blížila ke spodnímu okraji snímané oblasti. Je také zajištěná dostatečná velikost cílové oblasti

pro navigaci robotu. Nevýhodou tohoto řešení je nutnost zaklopení ramene robotu za kameru během navigace. Tato pozice byla zvolena jako nejvhodnější pro umístění kamery.

Další možností by bylo použití "složeného oka" - soustavy kamer, které by dokázaly pokrýt velkou oblast. Obrazy jednotlivých kamer by se potom překrývaly a cílová oblast by byla umístěna v obraze snímaném kamerou zaměřenou na rameno robotu ve sklopené pozici. Nevýhodou tohoto řešení je složitější algoritmus pro trasování objektů, který by musel řešit jejich přesun mezi jednotlivými snímanými oblastmi. Obraz všech kamer by nebylo možné snímat najednou, protože by došlo k prodloužení doby nutné k detekci objektů, proto by bylo nutné snímat obraz pouze z jedné kamery, která by byla volena dle pozice objektu. Předpokládá se, že snímané sousední oblasti by se částečně překrývaly. Pro toto řešení by také vzrostla doba prvního nalezení objektu, protože by bylo potřeba analyzovat postupně obraz ze všech kamer. Výhodou této metody je možnost snímání rozsáhlejší plochy včetně oblastí po stranách robotu.

4. Plánování trajektorie

4.1. Použitý hardware



Obrázek 4.1: Použitý hardware a směr toku dat.

Jako řídící jednotka je použit netbook firmy Asus Eee PC 901 (na obrázku 4.2):

- **Procesor:** Intel ATOM 1,6GHz FSB 533 Mhz,
- **Operační paměť:** 1 GB DDR2,
- **Pevný disk:** 4 GB SSD (solid state drive) systémový disk 8 GB SSD (solid state drive) disk pro data,
- Integrovaná kamera v displeji 1,3 megapixelu,
- **Napájení:** Li-Ion akumulátor 6600mAh Výdrž až 6 hodin,
- **Rozměry:** 225 x 170 x 20 - 33,8 mm,
- **Hmotnost:** 0,99 kg [13].

Jako operační systém byl zvolen openSuSE 12.4 pro svou dobrou spolupráci s řídící elektronikou, snadný přístup k V/V zařízením a možností volby instalovaných komponent,

což zajistilo zmenšení prostoru potřebného pro OS tak, aby jej bylo možné nainstalovat na primární disk /dev/sda. Velkou výhodou tohoto systému je existence předkompilovaného balíčku knihovny openCV.

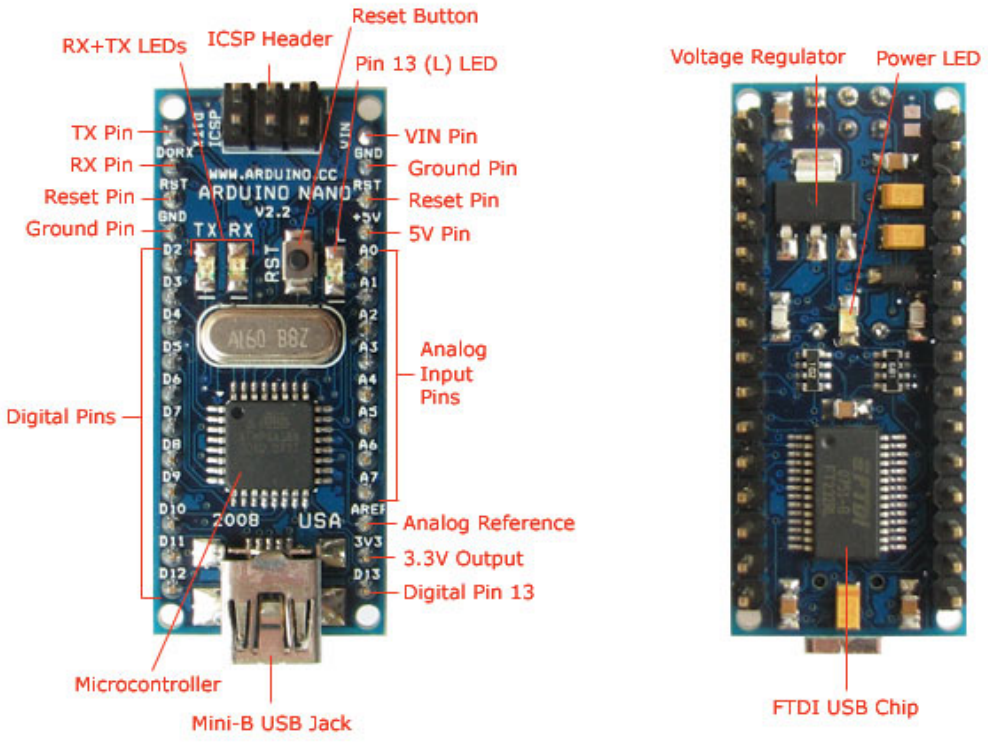
Obrázek 4.2: Řídící jednotka: Asus Eee PC 901.

Netbook sice obsahuje vlastní kameru, která je pevně zabudovaná v rámu obrazovky a při běhu robotu bude sklopena, tudíž ji nebylo možné pevně fixovat na určitý pohled. Z tohoto důvodu bylo potřeba použít kameru jinou, která umožníla získat potřebný pohled. Pro snímání obrazu tedy byla použita USB webová kamera firmy Logitech C210 s rozlišením 1.3Mpixelů (640x480px), která byla fixována přímo ke konstrukci robotu. Pro zvýšení rychlosti zpracování získaného obrazu je však použito pouze rozlišení 320x240px, které se ukázalo jako dostatečné pro detekci požadovaných objektů.

Pro řídící elektroniku je použit modul Arduino Nano 3.0 s mikrokontrolérem ATmega328. Modul je připojen pomocí čipu FTDI FT232RL k USB řídící jednotky. Zařízení se objeví v systému jako /dev/ttyUSB0 a /dev/ttyUSB1. Modul ramene se obvykle objeví jako zařízení /dev/ttyUSB0 a modul ramene jako /dev/ttyUSB1. Toto by bylo možné ověřit pomocí identifikačních zpráv. Pro komunikaci mezi řídící jednotkou a moduly byla navržena sada zpráv (přehled se nachází v příloze A). Délka zprávy je 8 Bytů a skládá se ze dvou částí: identifikátorové, jednoznačně určující příkaz a hodnotové, obsahující až 6 Bytů přenášené hodnoty. Každý vysílač/příjímač má svou vlastní sadu zpráv, ty jsou v této sadě navrženy tak, aby nemohly být navzájem zaměněny během interpretace. Přenosová rychlost byla zvolena nejvyšší možná 115200bd z důvodu potřeby rychlé synchronizace řídící jednotky s modulem motorů.

Signál z obou modulů Arduino Nano a webové kamery je sloučen do USB hubu zabudovaném v konstrukci robotu a signály jsou vedeny k řídící jednotce pouze pomocí jednoho kabelu. Usnadní se takto manipulace nebo případná výměna řídící jednotky a omezí se celková délka použitých kabelů, takže jsou všechny části mnohem přístupnější.

Použité části jsou zobrazeny na obrázku 4.1, včetně směru toku dat. US1-4 jsou ultrazvukové snímače vzdálenosti určené pro detekci objektů v blízkosti robotu. Modul Arduino1 obsahuje tlačítko určené pro odstartování časovače pro soutěž EuRobot. Kom-



Obrázek 4.3: Modul pro řízení elektroniky modul Arduino Nano[11].

presor a rameno zajišťují naložení předmětu. Tyto části a konstrukci robotu (na obrázku 4.4) popisuje ve své bakalářské práci kolega M. Studený.

Plánování trajektorie robotu bylo rozděleno do tří módů, podle zdroje řídícího signálu:

- manuální řízení pomocí bezdrátové klávesnice,
- plánovaní trajektorie na základě pozice robotu na mapě,
- plánování trajektorie na základě informace získané pomocí kamery.

4.2. Manuální řízení robotu

Tento mód je určen převážně pro pohyb robotu mimo hřiště, případně pro servis a ověření funkcí robotu. Zdrojem signálu pro řízení je (bezdrátová) klávesnice. Pohyb robotu je možné řídit numerickými klávesami. Pro pohyb vpřed/vzad slouží klávesy 8 a 2, pro rychlé otáčení kolem středové osy klávesy 4 a 6, pro pomalé otáčení klávesy 1, 3, 7, 9. Klávesou 5 je robot zastaven. Pohyb robotu je inicializován stisknutím klávesy a v tomto pohybu setrvává není-li změn. Bylo zvažováno setrvání robotu v pohybu pouze po dobu stisknutí klávesy, tento způsob se však ukázal méně efektivní z praktického hlediska především pro řízení na delší vzdálenosti. Klávesou 0 se odešle příkaz k naložení předmětu.

Obrázek 4.4: Konstrukce robotu.

4.3. Plánování trajektorie na zadanou pozici

Hřiště je rozděleno do tří oblastí a mapováno v paměti pomocí dvou dvourozměrných polí map[X][Y],map2[X][Y] o velikosti $310x210$ hodnot (velikost hřiště je $200x300$ $cm + 5$ cm z každé strany okraj. Každá hodnota představuje 1 cm na mapě). Pozice robotu na mapě je počítána od jeho středu a jeho rozměry jsou $30x30$ cm. X-ová souřadnice v grafickém zobrazení roste směrem doprava, Y-ová souřadnice roste směrem dolů.

První pole je **pole překážek**. Hodnotou -1 jsou v poli označeny zdi, určené herním plánem soutěže , které jsou v průběhu hry neměnné. Hodnotou -2 jsou označeny prázdné prostory kolem zdí, do kterých se robot kvůli své velikosti nemůže dostat a nebude potřeba jej do těchto míst navádět. Jedná se o cca 15 cm široký pás okolo všech statických zdí. Nulou jsou označena místa, která jsou volná a robot se v nich může pohybovat. Kladnou hodnotou jsou potom označeny dynamické překážky, na které robot během hry narazí. Tyto místa mají formu TTL (Time To Live) a jsou průběžně dekrementována. V praxi to znamená, že nalezená překážka (soupeřův robot) se po určitém čase přesune na jinou pozici, a prostor, který dříve blokovala, je opět volný.

Druhé pole označujeme jako **pole potenciálů**. Každá buňka udává potenciál pozice. Řízení robotu spočívá ve snaze získat pozici s co největším potenciálem. Toto pole je možné si představit jako pole s výškovými koeficienty, pozice s nejnižšími čísly by potom měly nejvyšší potenciál. Pokud by na takovu plochu byla položena kulička (jsou-li zanedbány setrvačné síly), pohybovala by se směrem dolů, dokud by neuvázla v lokálním (globálním) minimu. Tento model posloužil jako předloha pro plánování trajektorie pohybu robotu.

Trajektorie robotu je tedy plánována na základě informací o bezprostředním okolí robotu (cca 5 cm od jeho středu). V této oblasti je určeno lokální maximum a určen úhel odklonu od tohoto minima. Úhel odklonu je potřeba převézt do intervalu $< -\pi; \pi >$, toto je provedeno pomocí následujícího algoritmu:

```
dirAlpha = atan((double) (t_y - y) / (t_x - x));
t_alpha = (alpha - dirAlpha);
if (t_x < x)t_alpha += M_PI;

// optimalizace na nejmensi uhoel
if (t_alpha > M_PI) {
  t_alpha = t_alpha - 2 * M_PI;
}
if (t_alpha < -M_PI) {
  t_alpha = t_alpha + 2 * M_PI;
}
```

kde x a y jsou souřadnice robotu, t_x a t_y souřadnice bodu s nejvyšším potenciálem v okolí, $alpha$ je směr natočení robotu vůči mapě, $dirAlpha$ je odklon bodu s lokálním maximem vzhledem k mapě a pozici robotu, a t_alpha je odklon robotu od lokálního maxima. Na základě tohoto úhlu je určen vektor rychlosti pro jednotlivá kola. Směr pohybu je určen znaménkem odklonu, rychlost pohybu jeho velikostí. Pro úhel odklonu větší než $\frac{\pi}{4}$ $(90°)$ je zvolena vysoká rychlost otáčení (9 $pulzů/100ms$ pro režim bez kamery, 6 $pulzů/100ms$ pro režim s kamerou tak, aby nedocházelo k rozmazávání obrazu vlivem pohybu robotu). Pro úhel odklonu v intervalu $< \frac{\pi}{18}; \frac{\pi}{8} > (22,5° - 90°)$je zvolena rychlost poloviční tak, aby nedocházelo k "přejetí" středové oblasti. Pro úhel odklonu z intervalu $< 0; \frac{\pi}{8} > (0° - 22,5°)$ je zvolen pohyb v před maximální zadanou rychlostí (tj. $6 - 9$ $pulzů/100ms$).

4.3.1. Inverzní plánování trajektorie

Myšlenkou této optimalizace je použití rovnic 2.1 - 2.2 k inverznímu výpočtu rychlostí jednotlivých kol tak, aby bylo dosaženo požadované pozice, zadáním úhlu odklonu od lokálního maxima a rychlostí pohybu. Rovnice je možno upravit do následujícího tvaru:

$$dL = Sp + \frac{\alpha D}{2C_K} \; [pulzů/100ms] \tag{4.1}$$

$$dR = Sp - \frac{\alpha D}{2C_K} \; [pulzů/100ms] \tag{4.2}$$

kde α je úhel odklonu od lokálního minima a S_p je požadovaná rychlost robotu v $pulsech/100ms$. Protože největší nepřesnosti vznikají při změně směru otáčení kola, je požadováno, aby minimální rychlost každého kola zadaná touto optimalizací byla 0. Z této podmínky a rovnice 4.1 nebo 4.1 je možné určit operační rozsah pro tuto metodu:

$$\alpha_{max} = \left| \frac{2(dL - S_p)C_K}{D} \right| = \left| \frac{2(dR + S_p)C_K}{D} \right| \; [rad] \tag{4.3}$$

kde minimální rychlost pohybu je $dR = dL = 0$ $pulzů$, konstanta motorů $C_K = 0.25$ $cm/pulz$, rozteč kol je $D = 27$ cm a $S_p = 6$ $pulzu/100ms$ je minimální použitá rychlost v pulzech. Odtud lze určit maximální hodnotu pro operační rozsah:

$$\alpha_{max} = \left| \frac{2(S_p)C_K}{D} \right| = 0.111\ [rad] \tag{4.4}$$

což odpovídá 6.37°. Tento operační rozsah je však pro řízení robotu příliž úzký, protože při jeho použití dochází k překmitům mezi stavy určenými pro otáčení robotu, optimalizace má potom za následek výrazné zpomalení robotu. Abychom dosáhli požadovaného operačního rozsahu $< -\frac{\pi}{8}, \frac{\pi}{8} >$ bylo by potřeba zvolit vyšší rychlost pohybu, tu je možné odvodit z předchozí rovnice pro $\alpha_{max} = \frac{\pi}{8}$

$$S_p = \left| \frac{\alpha_{max} D}{2C_K} \right| \doteq 22\ [pulzů/100ms] \tag{4.5}$$

Tato rychlost je pro úlohu, na kterou je plánování trajektorie navrhována, příliš vysoká (především z důvodu malého manipulačního prostoru).

4.3.2. Výpočet potenciálového pole

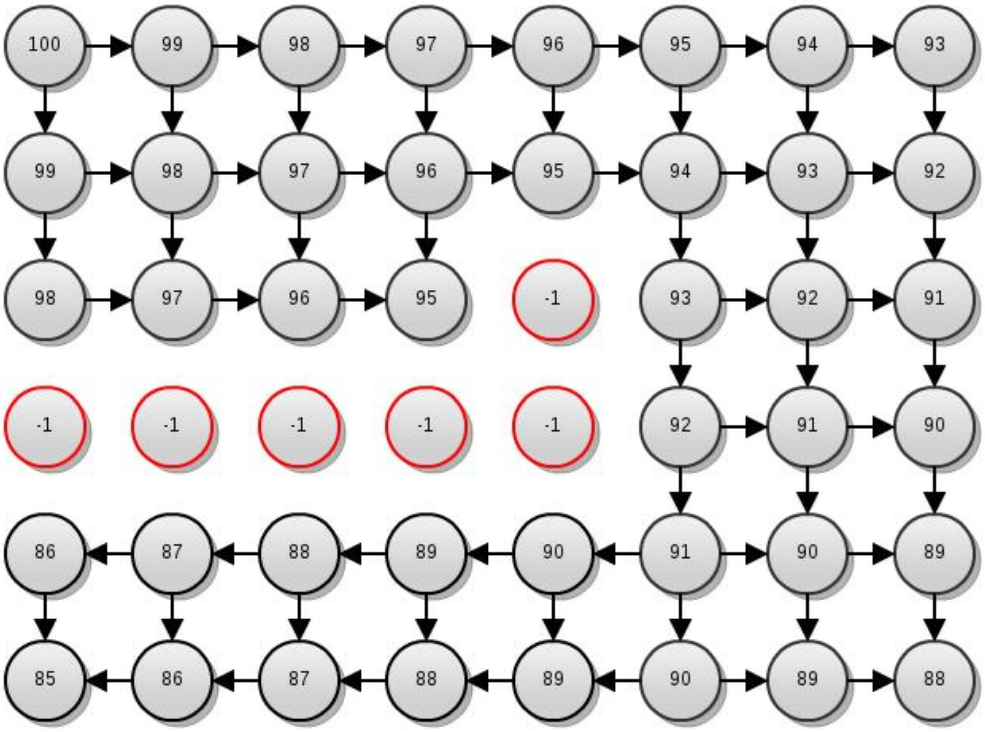
Pole potenciálu je neorientovaný graf s implicitnim zápisem hran. Dvě místa v poli jsou spojeny hranou právě tehdy, liší-li se jejich pozice právě jedné ze souřadnic (X nebo Y) právě o 1. Graf potom v rovinném zobrazení, kde X a Y jsou pozice bodu v cm, tvoří čtvercovou síť. Počáteční vrchol s pozicí $[0; 0]$ se nachází v levém horním rohu mapy hodnoty pozic potom rostou směrem doprava a dolů.

Při přepočítávání pole je nejprve potenciál všech pozic nastaven na nulu. Na pozici, kde je potřeba robot poslat, je nastaven maximální potenciál a z této pozice s klesající intenzitou šířen do okolí. Šíření je realizováno pomocí průchodu grafu do šířky podle následujícího algoritmu:

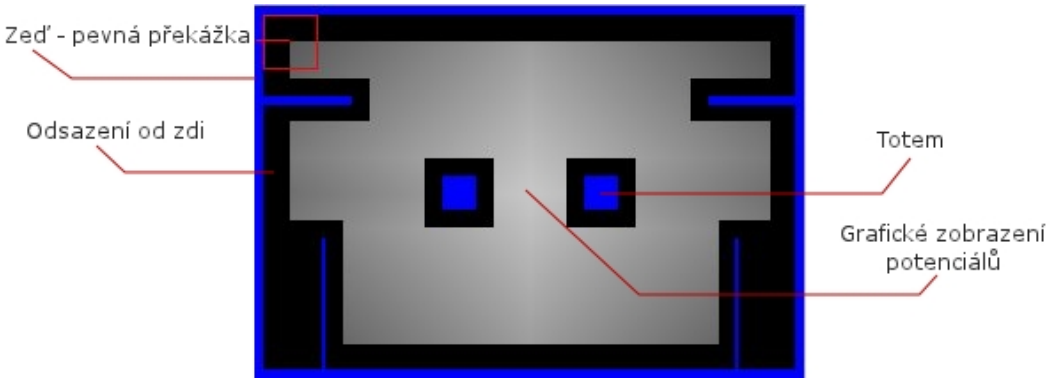
```
[1] Vlož do fronty požadovanou konečnou pozici robotu a maximální
    potenciál

[2] Je-li fronta prázdná ukonči šíření ( neexistují další neoznačené
    poice, kde by bylo možné se dostat )

[3] Vyber pozici a potenciál z fronty

[4] Je-li na pozici překážka ( zeď, odsazení od zdi, dočasná překážka )
    vrať se k bodu 2

[5] Má-li pozice již nastavený potenciál vrať se k bodu 2

[6] Nastav pozici potenciál, vlož sousední vrcholy a potenciál o 1 menší
    do fronty

[7] Vrať se k bodu 2
```

Nabízí se otázky typu: Dosáhne robot opravdu požadované pozice (za předpokladu, existence volné cesty) a dosáhne robot této pozice nejkratší možnou cestou?

Podmínkou dosažení požadované pozice je existence pouze jednoho lokálního minima v celém grafu. Pro každý vrchol s nenulovým potenciálem platí, že existuje nejméně jeden



Obrázek 4.5: Šíření potenciálu v potenciálovém poli (červeně jsou označeny překážky).



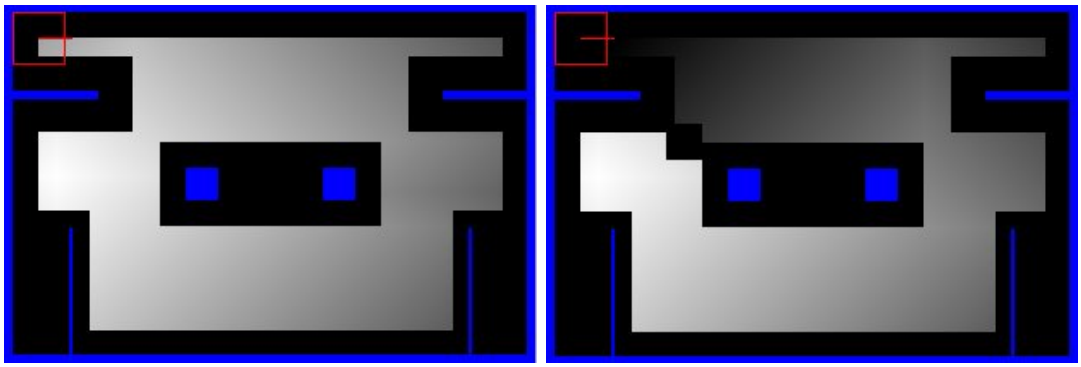
Obrázek 4.6: Mapa hrací plochy s potenciálovým polem a překážkami.

vrchol s hodnotou potenciálu právě o jedničku větší (vyplývá ze zápisu algoritmu). Vyjímkou z tohoto pravidla je pouze první vrchol, ze kterého bylo spuštěno šíření potenciálu, tedy výsledný vrchol, do kterého je potřeba se dostat. V grafu tedy nemohou vznikat lokální maxima.

Dosažení požadované pozice nejkratší možnou cestou docílíme použitím procházení grafu do šířky. Procházení do šířky zpracuje nejprve všechny vrcholy, n-tého potenciálu, poté všechny vrcholy $n - 1$ potenciálu, atd. Samotné šíření vypadá jako šířící se vlna a do každého vrcholu je možné se dostat nejbližší možnou cestou. Robot je poté navigován směrem k nejvyššímu potenciálu v jeho okolí.

```
void WorldMap::distribute(int t_x, int t_y, int t_value) {

    queInsert(t_x, t_y, t_value); // vlozeni pocatecniho bodu

    for (int i = 0; i < 310; i++)
        for (int j = 0; j < 210; j++)
            map2[i][j] = 0;

    while (!queEmpty()) {
        // vyjmuti prvku z fronty
        t_x = queGetX();
        t_y = queGetY();
        t_value = queGetValue();
        rmQueTop(); // odstraneni pozice z fronty

        if (t_value == 0)continue;
        if (map2[t_x][t_y] != 0)continue; // pozice jiz byla navstivena
        if (map[t_x][t_y] != 0)continue;  // na pozici je prekazka
        map2[t_x][t_y] = t_value;

        if (t_x < 5 || t_x > 305)continue;  // kontrola okraju mapy
        if (t_y < 5 || t_y > 205)continue;  // kontrola okraju mapy

        if (map2[t_x + 1][t_y] == 0)queInsert(t_x + 1, t_y, t_value - 1);
        if (map2[t_x - 1][t_y] == 0)queInsert(t_x - 1, t_y, t_value - 1);
        if (map2[t_x][t_y + 1] == 0)queInsert(t_x, t_y + 1, t_value - 1);
        if (map2[t_x][t_y - 1] == 0)queInsert(t_x, t_y - 1, t_value - 1);
    }
}
```

Grafické zobrazení mapy, včetně potenciálové vlny šířené od středu mapy, je na obrázku 4.6. Na mapě se nachází pevné zdi, které jsou označeny modře. Kolem zdí jsou černě označené odstupy. Tyto odstupy jsou nezbytné proto, že pozice robotu je počítána od jeho středu a do těchto míst se nemůže dostat. Spodní stěny lodě jsou dle pravidel soutěže zkoseny (viz obrázek 2.3), vzhledem k tomu, že robot do těchto míst nebude potřeba navigovat, je možné toto zkosení zanedbat. Na obrázku 4.7 je potom možné vidět, jak se změní potenciálové pole přidáním překážky, která znemožní použití původní cesty.



Obrázek 4.7: Změna potenciálového pole po přidání překážky.

4.3.3. Pohyb robotu po mapě

Pohyb robotu po mapě probíhá v pravoúhlých souřadnicích a kromě x, y souřadnic je potřeba ještě brát v úvahu úhel natočení robotu vůči mapě. Souřadnice robotu jsou vypočítávany z údajů o pohybů motorů získaných ze zabudovaných enkodérů pomocí rovnic 2.1 - 2.5. Získané hodnoty pak dosahují mnohem větších přesností než je centimetrová potenciálová mřížka, proto je pozice robotu zaokrouhlena na nejbližší pozici v této mřížce.

4.3.4. Optimalizace výpočtu směru pohybu

Protože nastavování rychlostí motorů (a odesílání informace o změně polohy) probíhá v časových intervalech $T_{vz} = 100ms$, je výpočet zatížen chybou, tedy směr pohybu je počítán z pozice, na které se již robot nenachází, a požadovaný směr je nastavován s již zmíněným zpožděním. Může proto dojít například k vjetí robotu do bezpečnostní oblasti v okolí zdí. Toto je možné eliminovat přidáním predikce příští pozice. Je známá pozice, ve které se robot nacházel při poslední změně pozice, a jsou také známy rychlosti obou motorů, proto je možné obdobným způsobem jako při přijetí zprávy o pohybu dopočítat pozici, ve které se bude robot s velkou pravděpodobností nacházet při příštím nastavování rychlostí motorů, a tak přesněji určit směr jeho dalšího pohybu. Touto optimalizací bylo možné zvýšit rychlost pohybu robotu po mapě až o polovinu při zachování stejné přesnosti navigace.

4.3.5. Využití metody

Tuto metodu by bylo možné upravit pro průzkum a mapování neznámého prostředí robotem. Počáteční stav by vypadal tak, že robot bude umístěn uprostřed prázdné potenciálové mapy. Z libovolného místa (nebo více míst) na okraji mapy se rozšíří potenciálová vlna. Robot by se vydá nejkratší možnou cestou ke zdroji této vlny. Ve chvíli, kdy narazí na překážku, zaznačí překážku do pole překážek. Vlna při dalším šíření už nemůže projít skrz objekt, je nucena jej obtéct a povede robot postupně kolem předmětu. Předmět bude zakreslen do pole. Tímto postupem je možné například navigovat robot ven z bludiště. Nevýhodou této metody je, že nezohledňuje výškové změny terénu, a proto by vyžadovala další úpravy pro pohyb ve složitějších prostorech (například ve vícepodlažních budovách).

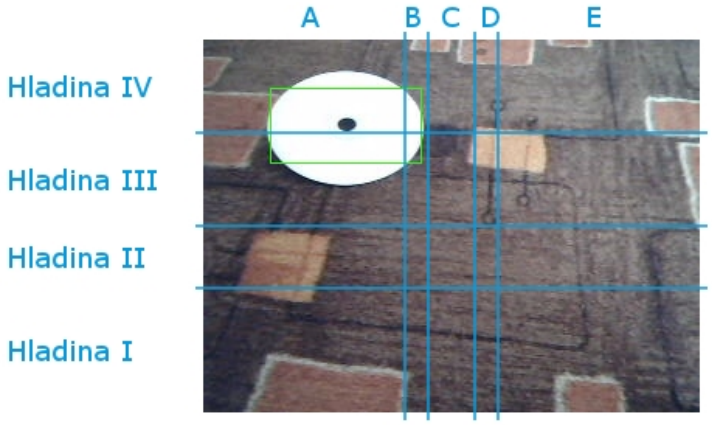
V praxi by bylo možné metodu využít například pro řízení provozu robotů v předdefinovaném koridoru. V tomto případě by byly kromě překážek definovány jízdní pruhy a přejezdy mezi nimi. Všichni roboti pohybující se v tomto koridoru by sdíleli stejnou mapu a každý by se hlásil svou pozicí, která by byla pro všechny roboty zaznamenána v této mapě jako dočasná překážka. Předpokládejme dva jízdní pruhy a dva roboty pohybující se různou rychlostí v koridoru. Pokud by byl pomalejší robot ve stejném jízdním pruhu před rychlejším, bude vytvářet za sebou potenciálový stín a rychlejší robot by byl potom automaticky sveden do druhého jízdního pruhu.

4.4. Plánování trajektorie robotu pomocí kamery

Úkolem tohoto způsobu plánování trajektorie je dosáhnout takové pozice robotu, aby se objekt detekovaný pomocí kamery nacházel ve vytyčené cílové oblasti. Cílovou oblastí rozumíme takovou pozici předmětu, ve které je možné jej uchopit pomocí mechanického

ramene robotu (při sklopení ramene robotu se detekovaný objekt nachází přímo pod ním, konkrétně pod přísavnými plochami ramene).

Detekovaný objekt je určen x,y souřadnicemi v obraze. Tyto souřadnice jsou středem detekované nalezeného objektu. Snímaný obraz je rozdělen do několika hladině označených I-IV a sektorů označených A-E (viz obrázek 4.8). Cílová oblast se nachází v hladine II a v sektoru C. Všechny oblasti, které s touto sousedí mají nastaveny nižší rychlost, robot je v nich přibržděn, aby došlo k přibrždění robotu ve chvíli, kdy se bude blížit cílové oblasti, a nedošlo k jejímu přejetí a následnému rozkmitání robotu mezi oblastmi B a D nebo většímu. Každá oblast má nastavený vlastní vektor pohybu, skládající se z rychlosti pohybu levého a pravého kola. Seznam všech oblastí včetně odpovídajících vektorů se nachází v příloze B.



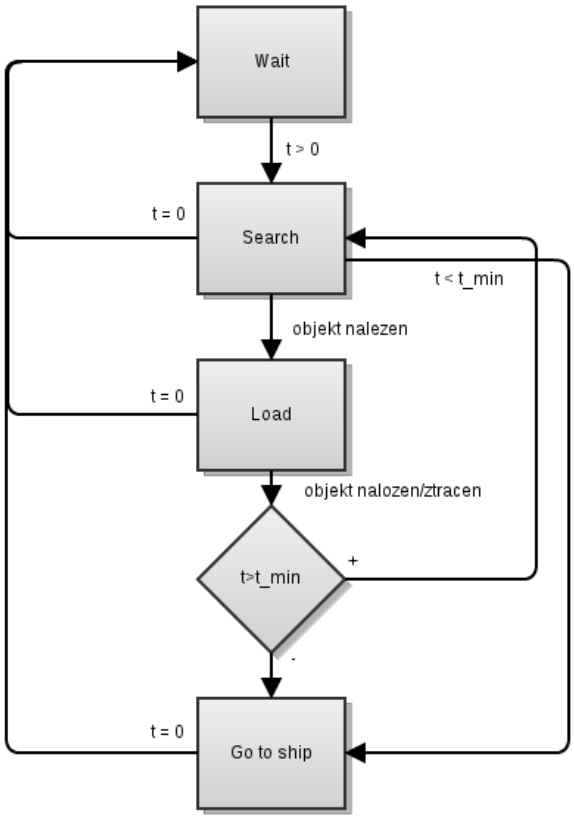
Obrázek 4.8: Snímek pořízený kamerou robotu s vyznačenými oblastmi.

4.5. Strategie pro EuRobot 2012 - Ostrov Pokladů

Mechanické provedení robotu bylo navrženo tak, aby umožňilo naložení co největšího množství počtu objektů typu mince. Toto by mělo vézt k ušetření času potřebného pro přesun mezi oblastmi mapy, kde se objekty nacházejí, a vykládací plochou (lodí). Byla tedy zvolená hladová strategie: robot se pokusí posbírat co největší množství předmětů typu mince (předpokládá se, že naložení jednoho objektu typu cihličky a jeho přemístění do lodě zabere stejný časový úsek jako naložení 3 objektů typu mince). Pro sběr objektů je vyhrazena doba hry zkrácená o potřebný čas (20 s) k přesunu z nejvzdálenější části hřiště k vykládací ploše. U vykládací plochy se nachází jeden objekt typu cihlička, robot je proto navigován tak, aby při své cestě tento objekt zatlačil dovnitř vykládací plochy. Robot ve vykládací ploše setrvá do ukončení zápasu, aby nedošlo k odcizení ukořistěných bodovaných objektů soupeřem.

4.6. Navržený algoritmus řízení

Algoritmus navržený pro řízení robotu se zakládá na stavovém automatu. Byly definovány stavy robotu, ve kterých se robot bude během plnění soutěžní úlohy nacházet a



Obrázek 4.9: Vývojový diagram návrhu řízení.

přechody mezi těmito stavy. Protože princip soutěže bývá pro každý ročník obdobný, je tento navržený algoritmus možné použít pro plnění více úloh, jen s minimální úpravou programu (pro každou úlohu je však potřeba provézt úpravu robotu pro sběr konkrétních předmětů). Vývojový diagram stavového automatu se nachází na obrázku 4.9.

4.6.1. Stavy a přechody mezi nimi

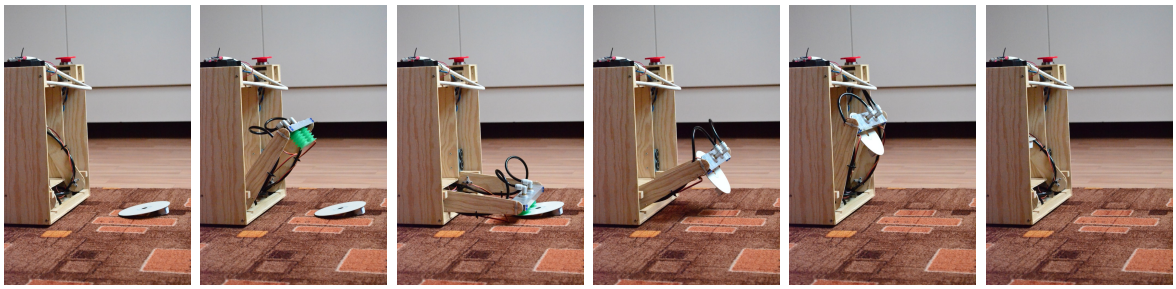
Wait je režim, ve kterém robot vyčkává odstartování zápasu, vyprší-li čas určený pro zápas (90 s), robot by měl opět skončit v tomto stavu.

- Je-li čas pro zápas $t > 0$ robot přechází do stavu **Search**.

Search je režim, ve kterém robot hledá předdefinované předměty na mapě. Není-li nalezen žádný předmět robot postupně projíždí kontrolní body, které by jej měly navézt na pozice na mapě, kde by se hledané předměty mohly nacházet (okolí totemů, soupeřova loď ...).

- Vyprší-li čas t určený pro zápas, robot se vrací do režimu **Wait**.
- Je-li $t < t_{min}$ robot robot přechází do režimu **Go To Ship**.
- Nalezne-li robot předmět, který hledá, přejde do režimu **Load**.

Load je režim, který je rozdělen do dvou etap. V první je robot navigován pomocí kamery tak, aby nalezený předmět umístil do cílové oblasti. Je-li dosaženo požadované pozice, navigace pomocí kamery je ukončena a předmět je ramenem robotu naložen. Ukázka nakládání objektu se nachází na obrázku 4.10.



Obrázek 4.10: USB kamera pro snímání obrazu.

- Vyprší-li čas t určený pro zápas, robot se vrací do režimu **Wait**.
- Nakládání dokončeno, robot má více času než t_{min}: přechází do režimu **Search**.
- Nakládání dokončeno, robot má méně než t_{min} času: přechází do režimu **Go To Ship**.

Go To Ship je režim pro návrat do vykládací oblasti. Robot je navigován na definovanou pozici na základě potenciálové mapy.

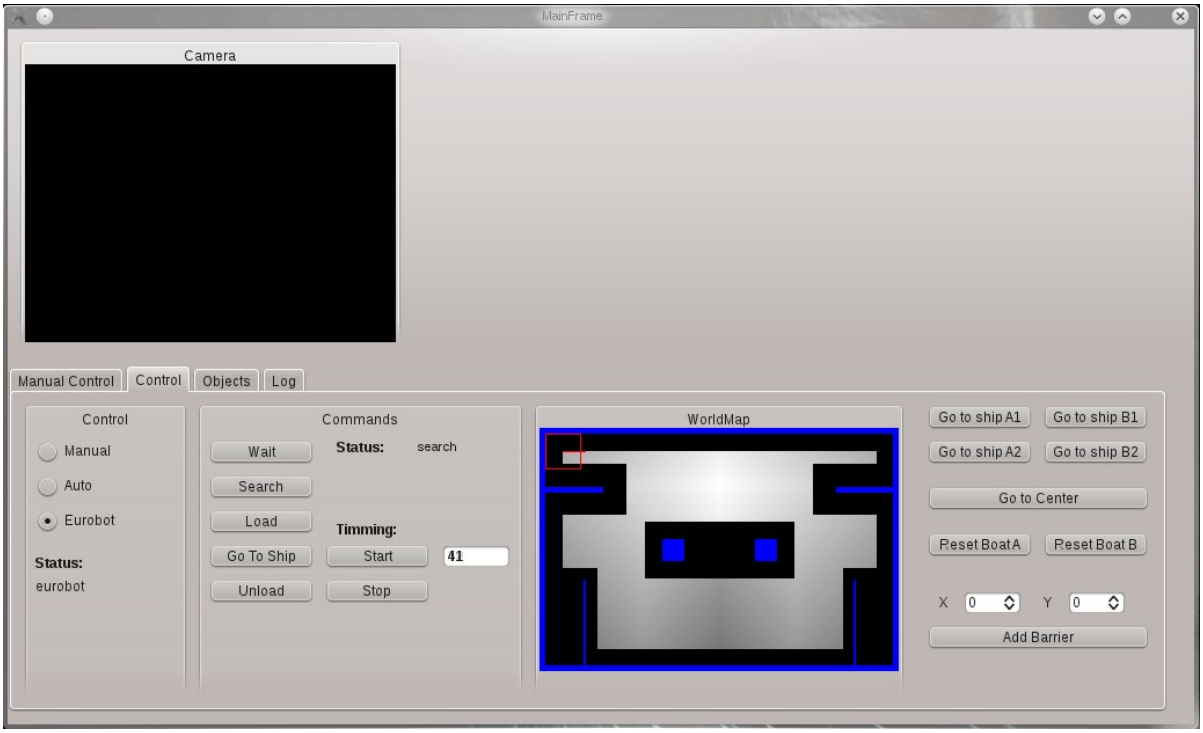
- Vyprší-li čas t určený pro zápas, robot se vrací do režimu **Wait**.

4.7. Ovládací rozhraní robotu

Pro ovládání robotu bylo vytvořeno pomocí QT knihovny grafické uživatelské rozhraní. To je rozděleno na dvě části: Horní část obsahuje obraz snímaný kamerou a volnou plochu připravenou pro přidání dalších informačních prvků, jako stav akumulátoru, hodnoty snímané ultrazvukovými senzory, a další.

Spodní panel obsahuje čtyři záložky:

- Záložka **Manual Control** obsahuje informace o manuálním řízení (rychlosti motorů).
- Záložka **Control** obsahuje ovládací prvky pro nastavení režimu, ve kterém robot pracuje (rám Control). Jako defaultní je zvolen manuální režim a v případě potřeby je možné do tohoto režimu přejít rychle pomocí klávesy Esc (robot okamžitě zastaví svou činnost a čeká další příkazy). Rám commands poskytuje informace o průběhu režimu EuRobot, pomocí tlačítek start/stop je možné tento režim dostartovat nebo zastavit. Rám WorldMap obsahuje mapu oblasti, ve které se robot pohybuje. V pravé části této záložky se nacházejí tlačítka pro automatický režim, pomocí kterých je možné poslat robot na předem definované pozice na mapě nebo umístit jej na některou z počátečních pozic). Protože doposud nebyly implementovány ultrazvukové senzory, je zde také ovládání pro simulaci nalezení překážky.



Obrázek 4.11: Ovládací rozhraní pro řízení robotu.

- Záložka **Objects** obsahuje seznam objektů nalezených pomocí kamery, včetně jejich rozměrů. Dále se zde nachází tlačítko na snímání obrazu, pomocí kterého je možné generovat obrazové vzory pro vytvoření klasifikátorů nebo jejich další ladění.
- Záložka **Log** obsahuje výpis krátkých zpráv o stavu robotu. Pomocí této záložky je možné zjistit například, jestli došlo ke správnému připojení externích modulů.

5. Závěr

V rámci této bakalářské práce byly navrženy a popsány metody a řídící algoritmy určené pro mobilní robot upravený pro soutěž EuRobot. Byl kladen důraz na obecný návrh těchto metod, které by bylo možné aplikovat na konkrétní řešení dle jednotlivých zadání soutěže. Navržená řešení byla prakticky otestována na úloze zadané pro rok 2012 - Ostrov pokladů.

Pro detekci objektů byl navržen postup pro vytvoření klasifikátoru ze zvolených vzorů. Dále byly navrženy a realizovány funkce, umožňující snímat obraz pomocí USB kamery, detekovat v něm objekty za běhu robotu a provádět jejich párování mezi jednotlivými, po sobě jdoucími snímky. Řídící program robotu byl navržen tak, aby bylo možné vytvářet nové vzory za běhu programu. Vznikl tak užitečný nástroj, jak pro vytvoření nového klasifikátoru, tak pro korekci existujících klasifikátorů, kterou je možné provézt doplněním pozitivních nebo negativních vzorů během ladění funkčnosti robotu.

Pohyb robotu je zaznamenáván na mapě, na základě které je možné plánovat trajektorii pohybu na určenou pozici. V této mapě jsou taky zaznamenány trvalé překážky nacházející se v prostředí, ve kterém se robot pohybuje, včetně požadovaného odstupu od těchto překážek. Dále jsou mapovány dočasné překážky, které jsou při pohybu robotu detekovány a jejichž pozice se může časem měnit. Protože ultrazvukové senzory nebyly doposud implementovány, je možné simulovat nalezení překážky v ovládacím rozhraní. Pro navržené algoritmy pohybu po mapě bylo také zhodnoceno jejich další využití v jiných aplikacích.

Jelikož konání soutěže EuRobot bylo v České Republice zrušeno, nebylo možné navrženého robota porovnat s jinými. Robot vytvořen v této bakalářské práci byl otestován v praxi a dá se říct, že jeho funkčnost z velké části splnila očekávání. Tato úloha byla velice zajímavá, poskytla prostor pro další řešení, proto bylo rozhodnuto ve vývoji robotu pokračovat a rozšířit jeho operační možnosti.

Literatura

[1] DEMEL,J.: *Grafy a jejich aplikace.* Praha, Academia, 2002, 257 s.
ISBN 80-200-0990-6

[2] McComb,G.: *Builder, Third Edition.* February 21, 2006, 770 s.
ISBN-13: 978-0071468930

[3] NOVÁK,P.: *Mobilní roboty - pohony, senzory, řízení.* BEN, 2005, 243 s.
ISBN 80-7300-141-1

[4] BRADSKI, G.; KAEHLER, A.: *Learning OpenCV: Computer Vision with the OpenCV Library.* OREILLY, 2008, Sebastopol, 577 s.
ISBN-13: 978-0596516130

[5] *OpenCV v2.4.0 documentation* [online] [cit. 26.5.2012]
Dostupné na: <http://opencv.itseez.com/modules/objdetect/doc/cascade_classification.html>

[6] *Objdetect module. Object Detection - OpenCV v2.4.0 documentation* [online] [cit. 26.5.2012]
Dostupné na: <http://opencv.itseez.com/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html#cascade-classifier>

[7] *AForge.NET :: Framework* [online] [cit. 13.3.2012]
Dostupné na: <http://www.aforgenet.com/framework>

[8] *AVXL - C++ Libraries for Computer Vision* [online] [cit. 23.3.2012]
Dostupné na: <http://vxl.sourceforge.net/>

[9] *Oficiální pravidla soutěže EuRobot* [online] [cit. 26.5.2012]
Dostupné na: <http://www.eurobot.org/>

[10] *Oficiální fórum soutěže EuRobot* [online] [cit. 26.5.2012]
Dostupné na: <http://www.planete-sciences.org/forums/viewtopic.php?f=2&t=15257/>

[11] *Arduino - ArduinoBoardNano* [online] [cit. 26.5.2012]
Dostupné na: <http://arduino.cc/en/Main/ArduinoBoardNano/>

[12] *Stavový diagram - Wikipedie* [online] [cit. 26.5.2012]
Dostupné na: <http://cs.wikipedia.org/wiki/Stavov%C3%BD_diagram#Stavov.C3.A9_automaty>

[13] *ASUS EEE PC 901 černý* [online] [cit. 26.5.2012]
Dostupné na: <http://www.alza.cz/asus-eee-pc-901-ceny-black-d93558.htm>

6. Seznam symbolů a zkratek

dL	dráha ujetá levým kolem robotu během jedné periody vzorkování.
dR	dráha ujetá pravým kolem robotu během jedné periody vzorkování.
$d\alpha_i$	změna úhlu natočení robotu během jedné periody vzorkování.
D	rozteč kol robotu.
dS_i	ujetá vzdálenost středu robotu během jedné periody vzorkování.
C_K	konstanta motorů pro převod na plošnou míru.
x_n	x-ová souřadnice robotu po n-tém kroku.
y_n	y-ová souřadnice robotu po n-tém kroku.
α_n	natočení robotu vzhledem k souřadnicové mřížce.
S_p	požadovaná rychlost robotu.

7. Seznam příloh

Příloha A: Seznam navržených zpráv.
Příloha B: Vektory pohybů pro řízení pomocí optického snímače.
Příloha C: Script pro generování klasifikátoru.

Příloha A: Seznam navržených zpráv

Žádost o identifikaci

Formát: IDXXXXXX;.
Vysílač: PC1.
Přijímač: Arduino1, Arduino2.
Popis: žádost o zaslání identifikace modulu, slouží k jejich rozlišení při startu programu.

Identifikace modulu

Formát: IAMxxXXX;.
Vysílač: Arduino1, Arduino2.
Přijímač: PC1.
Popis: identifikace modulu, xx - číslo modulu pevně určené v programu.

Nastavení rychlostí motorů

Formát: S$\pm xx \pm yy;$.
Vysílač: PC1.
Přijímač: Arduino1.
Popis: xx - rychlost levého motoru, yy - rychlost pravého motoru.

Zpráva o pohybu motorů

Formát: D$\pm xx \pm yy;$.
Vysílač: Arduino1.
Přijímač: PC1.
Popis: xx - zpráva o pohybu levého motoru, yy - zpráva o pohybu pravého motoru.

Zpráva z ultrazvukových senzorů

Formát: UxxyyXXX;.
Vysílač: Arduino2.
Přijímač: PC1.
Popis: xx - id senzoru, yy - vzdálenost překážky v cm.

Pokyn k naložení předmětu

Formát: LOADXXXX;.
Vysílač: PC1.
Přijímač: Arduino2.
Popis: pokyn ke sklopení ramene a naložení předmětu.

Nakládání ukončeno

Formát: LOADEDXX;.
Vysílač: Arduino2.
Přijímač: PC1.
Popis: informace o ukončení nakládání předmětu.

Odstartování

Formát: STARTXXX;.
Vysílač: Arduino1.
Přijímač: PC1.
Popis: informace o stisknutí startovacího tlačítka.

Příloha B: Vektory pohybů pro řízení pomocí optického snímače

Hladina	Sektor	Pozice [px]	Vektor pohybu [pulsů/100ms]
I	A	[0 130;160 240]	<-2;-1>
	B	[130 145;160 240]	<-1;0>
	C	[145 175;160 240]	<-2;-2>
	D	[175 180;160 240]	<0;-1>
	E	[180 240;160 240]	<-1;-2>
II	A	[0 130;120 160]	<-2;2>
	B	[130 145;120 160]	<-1;1>
	C	[145 175;120 160]	<0;0>
	D	[175 180;120 160]	<1;-1>
	E	[180 240;120 160]	<2;-2>
III	A	[0 130;60 120]	<1;2>
	B	[130 145;60 120]	<1;-1>
	C	[145 175;60 120]	<2;2>
	D	[175 180;60 120]	<-1;1>
	E	[180 240;60 120]	<2;1>
IV	A	[0 130;0 60]	<2;4>
	B	[130 145;0 60]	<2;4>
	C	[145 175;0 60]	<4;4>
	D	[175 180;0 60]	<4;2>
	E	[180 240;0 60]	<4;2>

Tabulka 7.1: Vektory pohybu v závislosti na pozici objektu ve snímaném obraze.

Příloha C: Script pro generování klasifikátoru

```
#!/bin/bash
#-------------------------------------------------------------------
# Libor Plucnar, 125596
# VUT - Brno
#


clear; # vycisteni obrazovky

WIDTH=40
HEIGHT=25

NUMPOS=45 # pocet positivnich prikladu
NUMNEG=257 # pocet negativnich prikladu
STAGES=14
SPLITS=1

MEM=1024

WEIGHTTRIMMING=0.5
MAXFALSEALARM=0.5

ALG=DAB # algoritmus DAB RAB RB GAB

VEC=data/createsamples.vec
CLAS=data/classifier

ls negatives/train | grep png > negatives/train/train.txt
ls negatives/train | grep jpg >> negatives/train/train.txt
echo "Negatives:"
cat negatives/train/train.txt | grep png -Ic
cat negatives/train/train.txt | grep jpg -Ic
echo "Positives:"
cat positives/train/train.txt | grep jpg -Ic

rm data -rf
rm log  -rf

mkdir data
mkdir data/classifier
mkdir log


LOG1=createsamples.log
LOG2=haartraining.log
LOG3=performance.log

echo "-----------------------------------------------------------";
echo "OPENCV_CREATESAMPLES:";
echo "-----------------------------------------------------------";
```

```
opencv_createsamples -info positives/train/train.txt -vec $VEC
                      -num $NUMPOS -w $WIDTH -h $HEIGHT > log/$LOG1


echo "-------------------------------------------------------------";
echo "OPENCV_HAARTRAINING:";
echo "-------------------------------------------------------------";
opencv_haartraining -data $CLAS -vec $VEC -bg negatives/train/train.txt
                    -npos $NUMPOS -nneg $NUMNEG -nstages $STAGES
                    -nsplits $SPLITS -mem $MEM -mode ALL -w $WIDTH -h $HEIGHT
                    -maxfalsealarm $MAXFALSEALARM
                    -weighttrimming $WEIGHTTRIMMING -bt $ALG  #> log/$LOG2


echo "-------------------------------------------------------------";
echo "OPENCV_PERFORMANCE:";
echo "-------------------------------------------------------------";
opencv_performance -data $CLAS -info positives/testing/testing.txt
                   -w $WIDTH -h $HEIGHT -rs $STAGES > log/$LOG3


echo "-------------------------------------------------------------";
echo "Testing:";
echo "-------------------------------------------------------------";

cat log/$LOG3
echo "-------------------------------------------------------------";
echo "Done!";
date;
echo "";
echo "";
```